UNIVERSIDAD
DE MÁLAGA

Departamento de Arquitectura de Computadores

TESIS DOCTORAL

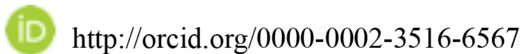
Genome Sequence Alignment in Processing-in-Memory Architectures

José Manuel Herruzo Ruiz

Enero de 2020

Dirigida por:
Dr. Oscar Plata González
Dra. Sonia González Navarro

UNIVERSIDAD
DE MÁLAGA



AUTOR: José Manuel Herruzo Ruiz

http://orcid.org/0000-0002-3516-6567

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga

Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional:
http://creativecommons.org/licenses/by-nc-nd/4.0/legalcode
Cualquier parte de esta obra se puede reproducir sin autorización
pero con el reconocimiento y atribución de los autores.
No se puede hacer uso comercial de la obra y no se puede alterar, transformar o hacer obras derivadas.

Esta Tesis Doctoral está depositada en el Repositorio Institucional de la Universidad de Málaga (RIUMA): riuma.uma.es

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

ANDALUCÍA TECH
Campus de Excelencia Internacional

Escuela de Doctorado

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA PARA OBTENER EL TÍTULO DE DOCTOR

D./Dña JOSÉ MANUEL HERRUZO RUIZ
Estudiante del programa de doctorado TECNOLOGÍAS INFORMÁTICAS de la Universidad de Málaga, autor/a de la tesis, presentada para la obtención del título de doctor por la Universidad de Málaga, titulada: GENOME SEQUENCE ALIGNMENT IN PROCESSING-IN-MEMORY ARCHITECTURES

Realizada bajo la tutorización de ELADIO GUTIÉRREZ CARRASCO y dirección de ÓSCAR PLATA GONZÁLEZ Y SONIA GONZÁLEZ NAVARRO (si tuviera varios directores deberá hacer constar el nombre de todos)

DECLARO QUE:

La tesis presentada es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, conforme al ordenamiento jurídico vigente (Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), modificado por la Ley 2/2019, de 1 de marzo.

Igualmente asumo, ante a la Universidad de Málaga y ante cualquier otra instancia, la responsabilidad que pudiera derivarse en caso de plagio de contenidos en la tesis presentada, conforme al ordenamiento jurídico vigente.

En Málaga, a 11 de JUNIO de 2020

Fdo.: JOSÉ MANUEL HERRUZO RUIZ

UNIVERSIDAD DE MÁLAGA

EFQM AENOR

Edificio Pabellón de Gobierno. Campus El Ejido.
29071
Tel.: 952 13 10 28 / 952 13 14 61 / 952 13 71 10
E-mail: doctorado@uma.es

UNIVERSIDAD
DE MÁLAGA

Dr. D. Oscar Plata González
Catedrático del Departamento de
Arquitectura de Computadores
Universidad de Málaga

Dra. Dña. Sonia González Navarro
Contratada Doctora del Departamento
de Arquitectura de Computadores
Universidad de Málaga

CERTIFICAN:

Que la memoria titulada "Genome Sequence Alignment in Processing-in-Memory Architectures" ha sido realizada por D. José Manuel Herruzo Ruiz bajo nuestra dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga, y constituye la Tesis que presenta para optar al grado de Doctor en Ingeniería Informática.

Málaga, Enero de 2020

Dr. D. Oscar Plata González
Codirector de la tesis

Dra. Dña. Sonia González Navarro
Codirectora de la tesis

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

A mi familia

UNIVERSIDAD
DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Agradecimientos

Durante estos años de trabajo he tenido el placer de convivir con muchas personas que, de una forma u otra, han formado parte de esta tesis y a las que me gustaría agradecer que hayan estado junto a mí en este periodo.

En primer lugar, quiero agradecer a mis directores Óscar Plata y Sonia González por darme esta oportunidad y por su apoyo y ayuda durante todo este tiempo. Sin duda han sido fundamentales para el desarrollo de esta tesis y sin ellos esto no hubiera sido posible. Muchas gracias.

Es necesario extender este agradecimiento a mis compañeros de laboratorio, con los que tanto tiempo he compartido y tanto he aprendido: Ricardo, Pedrero, Denisa, Villegas, Vilches, Gloria, Alex, Óscar, Esteban, Sergio, Fran, Antonio, Jesús, Bernabé, José Carlos, Iván, Andrés y Rubén. Tamben quiero dar las gracias a los demás compañeros de departamento, dentro de los cuales me gustaría destacar a Carmen, a los técnicos de Laboratorio Juanjo y Paco y a Inma, con quien he tenido el placer de compartir docencia. Hubiera sido imposible llegar a este punto sin vuestra ayuda.

I would also like to thank the people I met while in ESL at EPFL, who welcomed me there and helped me a lot when I was in a foreign place. Specially, I would like to mention David, Marina, Yasir, Luis, Tomás and Wellington. Thanks.

Nada de esto hubiera sido posible sin el inmenso apoyo incondicional de toda mi familia: A mis padres, mis hermanas Ana María e Isabel, mi padrino, mi madrina, mi tita Araceli, mis abuelas y todos mis tíos, por estar siempre ahí para todo lo que pueda necesitar y más.

UNIVERSIDAD DE MÁLAGA

También quiero agradecer a tantísimas personas maravillosas a las que puedo llamar amigos y a quienes no me merezco. Quiero destacar a algunos que llevan conmigo ya muchos años: Santiago, Josemi, Paco, Fausto, Fran, Tomás, Yaiza... y a algunos que han aparecido en esta etapa de mi vida: David, Salva, Amparo, Yoan, Pablo, Victor, Marco, Bea, María, Miriam, Patri... y muchos más que no he mencionado pero no por ello merecen menos agradecimiento.

Por supuesto, sería impensable no incluir en estos agradecimientos a la persona que más me ha aguantado estos últimos meses y que se ha convertido en una de las personas más importantes para mi. Carol, muchísimas gracias por estar siempre ahí, por apoyarme, darme siempre ánimos y por absolutamente todo. Gracias.

También es necesario nombrar las fuentes de financiación que han permitido llevar a cabo toda la investigación que ha dado como resultado esta tesis: Los proyectos de la Junta de Andalucía y el Gobierno de España y el plan propio de la Universidad de Málaga.

UNIVERSIDAD DE MÁLAGA

Abstract

Next generation workloads, especially for biomedical applications such as genome sequencing, have an astounding impact on the healthcare sector, for cancer research and drug discovery. The high demand for fast and low-cost genomic sequencing has pushed onward the rapid development of next-generation sequencing (NGS) technologies. Usually the first step in NGS corresponds to sequence alignment, where sequence reads must be aligned or compared to a genomic reference to identify regions of similarity.

When dealing with large reference genomes, great efforts were devoted to reduce memory requirements for sequence alignment. As a result, a set of alignment algorithms based on the FM-index structure have been developed. FM-index is well suited for fast exact matches of short reads to large reference genomes while keeping a small memory footprint.

As high-throughput sequencing systems produce a massive amount of data, the usage of high-performance technologies is of crucial importance to deal with the computational challenge.

A major performance bottleneck in high-performance computing (HPC) systems corresponds to the access patterns to memory due to the limited memory bandwidth available (memory wall problem [113]). Unfortunately, due to the data structure layout, the searching process using FM-index exhibits irregular memory access patterns. In addition, sequence aligners based on that index include support for inexact matching built on exact alignments, that causes the memory pattern to be even less predictable. These data access patterns cause a high cache miss rate on typical cache hierarchies present in HPC systems. Besides, it is common for the exact matching algorithm to be memory bound due to the low arithmetic intensity (ratio of the computation to the memory traffic). Each step of the algorithm accesses a section of the index that it is not known in advance, making the cache hierarchy difficult to exploit.

On the other hand, the amount of power used by HPC systems has been

UNIVERSIDAD DE MÁLAGA

non-stop increasing during the last decades, making energy consumption one of the most pressing and important problems for computer architecture. In 2018, the global data center energy demand was 198TWh, around 1% of the global energy demand [1]. This brings, in addition to the memory-wall problem, also the energy consumption into focus.

In this thesis, we address the problem of the low efficiency exhibited by the FM-index based sequence aligners on current HPC systems, due mainly to the memory bandwidth wall. We tackle this problem from different perspectives. Firstly, we analyze the FM-index exact matching algorithm. We consider different versions presented previously in the literature and propose a new organization of the data structure. With our new data structure we successfully minimize the demand for memory bandwidth and, therefore, improve the overall throughput and performance. After this analysis, we focus on improving even more the performance and energy efficiency of the mentioned application by developing a new Processing-In-Memory architecture. This architecture includes embedded in-order, energy efficient general purpose processors in the logic layer of a 3D-stacked memory. We analyze the results obtained after running several benchmarks and applications on the proposed architecture and on several conventional ones. Lastly, we continue with the architectural exploration focusing on a real, well-known and widely used sequence alignment application, Bowtie2. We conduct a detailed analysis of different architectural setups, comparing ARM low-energy cores with high performance ones. Experiments were carried out using a modified version of the gem5 architectural simulator in full-system mode for realistic results.

UNIVERSIDAD DE MÁLAGA

Contents

Agradecimientos **i**

Abstract **iii**

Contents **ix**

List of Figures **xiii**

List of Tables **xv**

1.- Introduction **1**

1.1 Memory Bound Application: Genomic Sequence Alignment 3

1.2 Data-centric Computing 4

1.3 Thesis motivation and research questions 4

2.- Background and Related Work **7**

2.1 Next-Generation-Sequencing or NGS 7

2.2 FM-index 8

2.2.1 Suffix Array 9

2.2.2 Burrows Wheeler Transform (BWT) 9

2.2.3 FM-index Data Structures 10

2.2.4 FM-index Search Algorithm 10

UNIVERSIDAD DE MÁLAGA

2.2.4.1 Count 10
2.2.4.2 Locate 11
2.2.5 Rank Query Implementations 11
2.2.6 Previous Works on Accelerating FM-index 12
2.3 Non-exact Matching using FM-index 12
2.3.1 Bowtie 13
2.3.2 BWA 13
2.4 3D-Stacked Memory Architectures 15
2.5 Intel Xeon Phi Knights Landing 15
2.5.1 Knights Landing Architecture 15
2.5.1.1 AVX-512 Instruction Set 15
2.5.1.2 KNL HBM Memory 16
2.5.1.3 Clustering Modes 17
2.6 PIM Architectures 18
2.6.1 Previous Work in PIM Architectures 19
2.7 Simulators 20
2.7.1 Gem5 21
2.7.2 ZSim 21
2.7.3 Ramulator 22
2.8 Roofline Model 22

3.- Split Bit-Vector Sampled FM-index Exact Matching 25

3.1 Analysis of FM-index variants 26
3.1.1 Basic FM-index 27
3.1.2 Sampled FM-index 28
3.1.3 K-step Sampled FM-index 30
3.1.4 Memory and Performance Analysis 31
3.1.4.1 Memory footprint 31

UNIVERSIDAD DE MÁLAGA

3.1.4.2 Memory access pattern 32
3.1.4.3 Search intensity 35
3.1.4.4 Throughput 36
3.1.5 Optimizing Throughput: Overlapped FM-index 37
3.2 Split Bit-Vector Sampled FM-index 39
3.2.1 Our approach: Split Bit-Vector Sampled FM-index 41
3.2.2 Memory footprint of our approach 42
3.2.3 Search Intensity and Throughput 42
3.3 Throughput Bounds Analysis 43
3.3.1 Instruction count 44
3.3.2 Throughput Bounds 45
3.4 Experimental Evaluation 47
3.4.1 Experimental Setup and Methodology 47
3.4.2 Throughput 48
3.4.3 Roofline Model 49
3.4.4 Comparison with Other Implementations 50
3.5 Related work 52
3.6 Conclusions 53

4.- FM-index Exact Matching Using Processing in Memory 55

4.1 Hardware Design and Simulation 56
4.2 Area and Power Consumption Estimation 58
4.2.1 PIM Setup Case 59
4.2.2 DDR Setup Case 60
4.3 Experimental Evaluation 61
4.3.1 Setup and Methodology 61
4.3.2 Results of the Benchmarks 63
4.3.3 FM-index Results 65

UNIVERSIDAD DE MÁLAGA

4.3.4 Roofline Model 67
4.3.5 Power efficiency 68
4.4 Related work 70
4.5 Conclusions 71

5.- Bowtie2 on Processing in Memory Architectures 73
5.1 Related work 74
5.2 Sequence Alignment Application: Bowtie2 75
5.3 Simulation Framework and Parameters for Architectural Exploration 77
5.3.1 Experimental Setup 77
5.3.2 Methodology 78
5.4 Results and Discussion 79
5.4.1 HBM2 vs DDR4 79
5.4.2 Near Compute HBM2 (no L2) vs DDR4 81
5.4.3 Performance-Energy Scaling with Core Count 82
5.4.4 Performance-Energy Scaling with Frequency 84
5.4.5 Comparison to Intel Xeon Phi KNL 85
5.5 Conclusions 88

6.- Conclusions 89
6.1 Future work 93

Appendices 95

A.- Random Memory Access Benchmark 95

B.- Resumen en español 101
B.1 Motivación y Temas de Investigación 102
B.2 Alineamiento de Secuencias con FM-index 103

UNIVERSIDAD DE MÁLAGA

B.2.1 Análisis de FM-index 103
B.2.1.1 Patrón de Accesos a Memoria 104
B.2.1.2 Rendimiento 105
B.2.1.3 K-Step Sampled FM-index 106
B.2.2 Split Bit-Vector k-Step Sampled FM-index 107
B.2.3 Evaluación Experimental 109
B.3 FM-index y Procesado en Memoria 111
B.3.1 Diseño de Arquitecturas y Simulación 111
B.3.2 Estimación de Area y Consumo Energético 111
B.3.3 Evaluación Experimental 113
B.4 Bowtie2 y Procesado en Memoria 115
B.4.1 Bowtie2: Aplicación Completa de Alineamiento de Secuencias 117
B.4.2 Entorno de Simulación Arquitectural 117
B.4.3 Resultados y Discusión 119
B.5 Conclusiones 120

Bibliography **125**

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

List of Figures

2.1 BWT M Matrix 9
2.2 Exact matching count operation 10
2.3 Bowtie: Inexact matching diagram 14
2.4 STREAM Benchmark results 16
2.5 PIM Architecture diagram. 19
3.1 Basic backward search algorithm based on FM-index 27
3.2 a LFM-chain, where u is either sp or ep 28
3.3 Basic and sampled FM-index data structures 29
3.4 k-step FM-index (k=2) 31
3.5 Fraction of $LF(c, ep)$ calls that access the same cache block as companion $LF(c, sp)$ calls for different text and query sizes of 200 and 400 symbols 33
3.6 Backward search timing model, where L_x represents latencies for the different phases. 36
3.7 Backward match algorithm overlapping N_q queries 38
3.8 Backward search timing model with N_q overlapped queries 39
3.9 FM-index data structures 40
3.10 Sampled, k-Step and Bit-vector FM-Index cache mapping. 41
3.11 Throughput for different FM-index versions 48
3.12 Broadwell roofline model 50
3.13 Skylake roofline model 51

UNIVERSIDAD DE MÁLAGA

3.14 KNL roofline model 51
4.1 Processor power consumption for each architecture 59
4.2 STREAM [80] and RANDOM memory bandwidth results for several architectures 62
4.3 Benchmark memory bandwidth in memory blocks per second for different arithmetic intensities. (*) Marked setups use Zsim memory model 64
4.4 Benchmark memory bandwidth for different arithmetic intensities. (*) Marked setups use Zsim memory model 65
4.5 Benchmark operations per second for different arithmetic intensities. (*) Marked setups use Zsim memory model 66
4.6 FM-index throughput for several architectures 67
4.7 i7-8700 Roofline model 68
4.8 DDR3 Roofline model 69
4.9 DDR4 Roofline model 69
4.10 PIM roofline models 70
4.11 RANDOM memory bandwidth and LFOPs per Joule for different architectures 71
5.1 Application phases for (a) FM-index and (b) Bowtie2 76
5.2 Performance benefit of HBM2 vs DDR4 with same cache hierarchy 80
5.3 Energy benefit of HBM2 vs DDR4 with same cache hierarchy . . . 80
5.4 Performance benefit of HBM2 with no L2 vs DDR4 with L2 81
5.5 Energy benefit of HBM2 with no L2 vs DDR4 with L2 82
5.6 Performance scaling at 2GHz 83
5.7 Energy scaling at 2GHz 84
5.8 Performance scaling at 1GHz 85
5.9 Energy scaling at 1GHz 86
5.10 Performance scaling at 1.5GHz for HBM2 87
5.11 Bowtie2 execution times on Xeon Phi vs different ARM configurations 87

UNIVERSIDAD DE MÁLAGA

A.1 RANDOM benchmark blocks diagram 96
A.2 RANDOM benchmark 97
A.3 Memory bandwidth for Broadwell, Skylake and KNL obtained with the RANDOM benchmark for various values of C 97
A.4 Comparison of RANDOM memory latencies 98
A.5 RANDOM benchmark timing model 99
B.1 Algoritmo básico de búsqueda *BS* basado en FM-index. 104
B.2 Modelos temporales para algoritmos BS (izquierda) y OBS (derecha), donde L_x representa latencias. 105
B.3 Estructuras de datos Sampled FM-index 106
B.4 Estructura de datos FM-index 107
B.5 Límites teóricos y rendimiento para diferentes versiones de FM-index. 109
B.6 Modelo *roofline* para Knights Landing (KNL). 110
B.7 Modelo *roofline* para Skylake (SKL). 110
B.8 Diagrama de arquitectura PIM 112
B.9 Consumo de energía por procesador 113
B.10 Benchmark STREAM y RANDOM para distintas arquitecturas . . 114
B.11 Rendimiento de FM-index para varias arquitecturas 115
B.12 Modelo Roofline para arquitectura PIM 116
B.13 Ancho de banda con accesos aleatorios y operaciones LF por julio de energía 116
B.14 Fases de los algoritmos de (a) FM-index y (b) Bowtie2 118
B.15 Mejora de rendimiento de HBM2 vs DDR4 119
B.16 Mejora de rendimiento de HBM2 sin L2 vs DDR4 con L2 120
B.17 Escalado de rendimiento a 1GHz 121
B.18 Escalado de energía a 1GHz 122

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

List of Tables

3.1 Basic and sampled FM-index properties (B (GB) stands for bytes (Giga bytes), CB for cache blocks and P for padding) 34

3.2 Split bit-vector k-step sampled FM-index parameters, for $k = 2$ . . 42

3.3 Features of processors used in the evaluation 44

3.4 Instruction count and computation latency per $sLF_k()$ call 44

3.5 Throughput limits for Broadwell and Skylake 45

3.6 Throughput limits for KNL 46

3.7 Branch predictor misses 49

3.8 Match operation performance 50

4.1 Hardware simulation setups summary 57

5.1 LLC Sizes and scaling with number of cores. 79

B.1 Propiedades del algoritmo de búsqueda para diferentes versiones de FM-index (p.ej. $k1$-$32SFM$ corresponde a k-SFM con $k = 1$ y $d = 32$). 108

B.2 Arquitecuras hardware simuladas 112

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

1 Introduction

In recent years there has been tremendous growth in data generation and in the number of applications that process large amounts of data, both in the scientific and industrial context (for example, meteorology, genomics, environmental sciences ...), as well as in social context (for example, social networks, finance, searches on Internet...). Conventional parallel programming architectures and models behave in an inefficient way when they execute this type of applications, as it usually involves the movement of a large amount of data between the different units of the system. This is mainly because the data must migrate from some level of the memory hierarchy to the computing units where they are processed. To solve this problem, some authors have designed new data-centric systems, characterized by performing the processing in the same place where the data is stored, minimizing data migration and, therefore, improving performance, energy consumption and reliability.

Improvements in memory systems performance is keeping far behind the development of computing systems for many years. This fact, together with the previous mentioned increase in the amount and importance of data intensive applications, has lead to the advent of the memory-wall problem [113]. In the literature, the term memory-wall is used to refer to the system bottleneck between memory and processors, affecting both performance and energy consumption of modern computing systems, specifically for those applications which do not take enough advantage from traditional cache systems.

This bottleneck is due to the high cost of data movement. For traditional memory systems, each memory operation requires a slow and energy consuming procedure in order to retrieve the data from main memory, trasfer it to the CPU, process it in the CPU and, if necessary, write back the output data to memory.

UNIVERSIDAD DE MÁLAGA

Specifically, each memory access follows the next steps: 1) the CPU issues a request to the memory controller; 2) the memory controller sends a series of commands to the memory DRAM module across the off-chip bus; 3) the required data and some more data around it (cache line) is read from memory, sent to the CPU and stored there, passing by the cache memory hierarchy and being replicated several times. Only after this process, the data can be processed and used for computing. In addition to this, the data replicated in cache memories are not always used again, specially when running some specific application workloads with non-uniform data access patterns or with low or no data reuse.

On the other hand, power consumption is becoming a pressing problem for High Performance Computing (HPC) systems. The amount of energy consumed by HPC systems is non-stop increasing during the last years, reaching in 2018 the 1% of the global energy demand [1]. The increase in power consumption leads to an ineluctable increase in heat generation, creating more challenges for HPC. Data movement also supposes an important part of the energy consumption, making any optimization of the memory system a priority in current HPC research.

This thesis is organized as follows. In the rest of Chapter 1 we describe briefly Processing in Memory (PIM) and Near Data Processing (NDP) architectures and memory bound applications, such as sequence alignment, and provide a motivation for this thesis. Chapter 2 includes a description of some background and related work necessary to understand the state of the art and introduces the tools and frameworks used in the elaboration of this thesis. In Chapter 3, we analyze the FM-index exact matching algorithm and data structures, the current variants of the FM-index in the literature and perform an in-deep analysis of the performance of FM-index focusing mainly on the usage of the memory. After this analysis we present our new FM-index data structure and algorithm. Chapter 4 presents a new PIM architecture able to improve the performance of applications with random memory accesses and no data reuse, with special focus on the backward search algorithm based on FM-index. Chapter 5 presents another data-centric architecture, focusing on the detailed analysis of the simulations and the evaluated application, Bowtie2, a widely used sequence aligner. Lastly, Chapter 6 presents the conclusions of this thesis.

UNIVERSIDAD DE MÁLAGA

1.1 Memory Bound Application: Genomic Sequence Alignment

The high demand for fast and low-cost genomic sequencing has pushed onward the rapid development of next-generation sequencing (NGS) technologies. As a result, a number of high-throughput sequencing systems have appeared in industry, including those from Illumina, Roche 454, Life Technologies and Pacific Biosciences. These systems are able to produce huge amounts of short reads (in the order of giga base-pairs) per day of operation. For instance, the Illumina NovaSeq 6000 sequencing system is able to produce up to 20 billion reads of 150 base pairs (bp) in less than two days. This represents up to 6 terabits of data which have to be processed as fast as possible.

Usually the first step in NGS corresponds to sequence alignment, where sequence reads must be aligned or compared to a genomic reference to identify regions of similarity [71]. Most popular alignment methods are based on two types of index structures: suffix trees and variants, and hash tables.

When dealing with large reference genomes, great efforts were devoted to reduce memory requirements for sequence alignment. As a result, a set of alignment algorithms based on the FM-index structure have been developed [33]. The FM-index uses the Burrows-Wheeler transform (BWT), a method for rearranging a character string that is useful for data compression [21]. FM-index is well suited for fast exact matches of short reads to large reference genomes while keeping a small memory footprint. Many efficient sequence aligners are based on FM-index, such as Bowtie [65], Bowtie2 [64], BWA [70] (BWA-SW [69] for long reads), SOAP2 [72] and BWT-SW [63].

Due to the data structure layout, the searching process using FM-index algorithms exhibits irregular memory access patterns. In addition, sequence aligners based on FM-index include support for inexact matching based on exact alignment algorithms. This causes the memory pattern to be even less predictable. Random memory access patterns cause a high cache miss rate on typical cache hierarchies of multicore processors because each step of the algorithm accesses a section of the index that it is not known in advance, making the cache hierarchy difficult to exploit. Moreover, it is common for the sequence alignment algorithm to be memory bound due to the low arithmetic intensity (ratio of the computation to the memory traffic).

UNIVERSIDAD DE MÁLAGA

1.2 Data-centric Computing

The concept of data-centric computing is known as Near-Data Processing (NDP) [13]. NDP is not a new concept. By the end of the last century there was a lot of research activity in similar lines, such as PIM (Processing-In-Memory) [107], whose goal was to perform part of the data processing in specialized units located in the physical memory itself. However, this type of works did not become relevant enough as a result of significant difficulties related to their commercial adoption (basically, manufacturing costs). Nevertheless, there has been recently a resurgence of the concept of carrying the processing into memory. This appears as a consequence of the growing importance of big-data applications [25], and the evolution of new manufacturing technologies, such as Through-Silicon-Vias (TSVs) [83], that have allowed the development of 3D-stacked DRAM memory[16].

These advances have led to the development of several relevant systems in this field. The most significant would be the Hybrid Memory Cube (HMC) [81] developed by Micron in collaboration with others manufacturers like Samsung, ARM, etc., and the High Bandwidth Memory (HBM) [4] powered by AMD, which has been already implemented in several GPUs. Significant parts of data movement between memory and processors can be avoided using those modern 3D-stacked memory architectures which move the computation to the logic layer underneath memory cubes. Applications running on this logic layer can take a significant advantage from the huge memory bandwidth and low latency, as well as significantly reduce the power consumption and increase the performance for memory-bound applications.

1.3 Thesis motivation and research questions

The mentioned memory-wall bottleneck and the growing importance of scientific memory-bound applications, specifically in the field of genomic sequence alignment in bioinformatics based on FM-index, create a problem which should be addressed. We think that this problem must be approached from two points of view:

- The algorithmic side of the problem, analysing and improving a well-known sequence alignment algorithm based on FM-index.

- The architectural side of the problem, working on the development of new

UNIVERSIDAD DE MÁLAGA

data-centric systems which could help to reduce the impact of the memory-wall and power consumption problems.

As a result of this approach, we define the general goal of this thesis project as the design of novel architectural and algorithmic solutions to overcome the physical memory barrier. We focus on applications and workloads that process massive amounts of data, and, specifically, those related to genome sequence alignment. Moreover, we believe that nowadays and, in the future, is essential to focus on energy efficiency, a factor whose importance is increasing every day and one of the most key limiting factors in HPC.

Our main contributions in this thesis and the related publications, which intend to answer the previous research questions are the following:

- We analyze the behavior of memory-bound applications with random memory access, which are not able to take advantage of modern memory architectures. Specifically, we analyze FM-index, an exact matching algorithm widely used in sequence alignment.
- After obtaining a deeper knowledge of the memory-wall bottleneck, we work in a new FM-index version, reducing the memory bandwidth and computing power it uses, improving significantly the performance. Memory bandwidth use is reduced around a 75% per operation and performance improves up to 135% when compared with previous implementations.
- We propose a new energy-efficient architecture based on a 3D-stacked memory cube, adding some low-power in-order cores to the logic layer. This architecture, oriented to random memory access, is able to provide high memory bandwidth with low latency and an important energy saving , increasing efficiency up to 40 times.
- We propose a second novel architecture, using HBM2 and we use Bowtie2 as test application. We simulate the architecture using a full-system architectural simulator, obtaining a more realistic view of the problem and how this architectural exploration can help to solve it. This architecture makes Bowtie2 performance improve up to 68% with an energy benefit up to 71%.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

2 Background and Related Work

In this chapter we present some background related to FM-index. In addition, we introduce the tools and architectures that we use or explore in this thesis in order to address the issues brought up in the previous chapter. More specifically, section 2.2 is devoted to revise FM-index structures and algorithm, whereas section 2.3 introduces other alignment applications. Section 2.4 is dedicated to review the current 3D-stacked memory systems. In section 2.5 the characteristics of the Intel Xeon Phi Knight Landing (KNL) architecture, that includes on-package 3D-stacked memory, are presented. Section 2.6 summarizes the features and the state-of-the-art of Processing-in-Memory (PIM) architectures. And lastly, section 2.7 and section 2.8 present some of the tools used in this thesis to analyze some of our proposals.

2.1 Next-Generation-Sequencing or NGS

NGS is the name given to a set of new methods and techniques for high speed and low cost DNA sequencing. It is also known as second generation sequencing or massively parallel sequencing. NGS applications are having a great impact on numerous fields, like bioinformatics, cancer research or food microbiology [17, 94, 118].

An example of a Next Generation Sequencing process workflow can be the following steps [27]:

UNIVERSIDAD DE MÁLAGA

1. Extraction of genomic DNA from samples.

2. DNA shearing with a method of choice.

3. Ligation of specific reporters.

4. Library selection and purification.

5. Library amplification using PCR.

6. Sequencing.

7. Sequence alignment and assembly of gene annotations.

8. Bioinformatics analysis.

Most of the mentioned steps (1-6) are mainly conducted in biology or biochemistry, staying out of the scope of this thesis. In Sequence Alignment, the 7th step and the first strongly related to High Performance Computing, a sequence read is aligned or checked against a genomics reference for regions of similarity. This process usually supports non-exact matching, due to errors in the sequencing process or differences between samples.

Several sequencing applications are based on FM-index, a compressed full-text index which we are studying in this thesis. We focus on FM-index algorithm because combines indexing and efficient storage with no significant slowdown in query time compared to full-text indices and because exhibits random memory access and is a memory-bound problem fulfilling one the goals of this thesis.

2.2 FM-index

FM-index is a data structure that allows fast substring searches over large texts [33]. This data structure is based on the Burrows-Wheeler transform (BWT) [21], which rearranges a text string into a form that is easier to compress. In the following subsections we show an analysis of some basic definitions concerning FM-index data structures and search algorithms.

In the rest of the thesis, brackets are used to specify an entry in an array (e.g., $A[k]$ is k-th entry of the array A), and a character or a substring in a string (e.g., $A[k]$ is the character at position k in the string A, and $A[k..r]$ is the substring from position k to position r in A).

UNIVERSIDAD DE MÁLAGA

2.2.1 Suffix Array

Let $T[1..n]$ be a character string drawn from an alphabet Σ having σ symbols. The suffix array $SA[1,...,n]$ of T is an array containing the starting positions of all suffixes of T in lexicographical order [79]. For instance, if $T = [gtaata]$ then $SA = [6,3,4,1,5,2]$. The suffix array requires $n \lceil log_2 n \rceil$ bits, where n is the length of T. The suffix array can be used as an index to locate every occurrence of a pattern. Searching in a suffix array can be done using a binary search algorithm in $log_2 n$ steps.

2.2.2 Burrows Wheeler Transform (BWT)

The BWT is a permutation of a character string. Originally it was used in text compression algorithms, but it has other applications such as large text indexing.

The $BWT[1..n{+}1]$ of a n-character string T is another string obtained as follows:

1. Append to the end of the original text T the symbol $, which is lexicographically smaller than any symbol in Σ.

2. Form a conceptual $(n{+}1){\times}(n{+}1)$ matrix M whose rows are the cyclic shifts of T\$ sorted in lexicographical order. This matrix M is shown in Figure 2.1.

3. The last column of matrix M is the BWT of T, denoted by L. This last column can be used to recover the original text T.

The suffix array of T\$ contains the starting positions in T of every row of the matrix M.

F						**L**
$	G	T	A	A	T	A
A	$	G	T	A	A	T
A	A	T	A	$	G	T
A	T	A	$	G	T	A
G	T	A	A	T	A	$
T	A	$	G	T	A	A
T	A	A	T	A	$	G

Figure 2.1: Matrix M used for the BWT generation of the text GTAATA$

UNIVERSIDAD DE MÁLAGA

2.2.3 FM-index Data Structures

FM-index combines indexing and efficient storage in such a way that no significant slowdown in query time is achieved compared to full-text indices. FM-index is composed of two data structures derived from L (BWT of T): the C array and the Occ matrix. The C array stores in $C[c]$ the number of occurrences in L of the symbols lexicographical smaller than c. $Occ[c,i]$ contains the number of occurrences of the symbol c in the prefix $L[1..i]$, being $1 \leq i \leq n+1$. C and Occ can be used to efficiently locate each occurrence of a character pattern within T.

The FM-index was designed as a compressed structure such that the index size can be smaller than the original text. However, in the context of sequence alignment, it is usually not compressed in order to achieve better performance [71].

2.2.4 FM-index Search Algorithm

Given a pattern $Q[1..p]$, the FM-index allows to find all occurrences of Q in the text T [33]. The search process consists of two operations denoted as *Count* and *Locate*.

2.2.4.1 Count

Count operation (shown in Figure 2.2) is a rank query process that calculates the number of occurrences of Q in T by identifying the first and last rows of matrix M (see sec. 2.2.2) prefixed by the query Q. This operation is explained in more detail in the FM-index analysis presented in chapter 3.

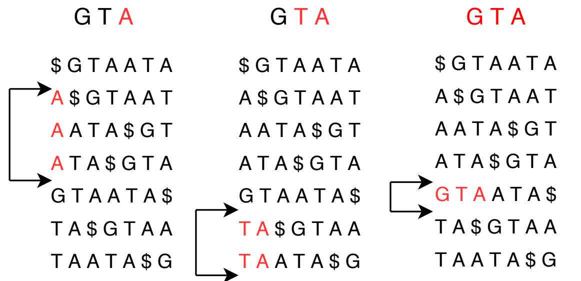


Figure 2.2: Exact matching count operation

UNIVERSIDAD DE MÁLAGA

2.2.4.2 Locate

The locate operation uses the indexes of the rows obtained by the count operation to access the suffix array, where it finds the position of every occurrence of Q in the text T.

The suffix array is usually a very large compressed data structure. However, its size for the human genome is about 12 GB (3 gigabases x 4 bytes), so it can be stored without compression in modern systems. In this way, the *Locate* operation turns out to be very simple, as it only requires an access to the suffix array. For this reason, our effort focuses on improving the performance of the *Count* operation.

2.2.5 Rank Query Implementations

To speed-up the *Count* process, FM-index uses the *Occ* matrix as a look-up table [33]. The main drawback of this solution is the large memory footprint of *Occ*. It is a matrix with σ rows and n+1 columns. Hence, its footprint is:

$$Fp(Occ) = \sigma \times (n + 1) \times Fp(Occ_{entry}), \qquad (2.1)$$

where the size of Occ_{entry} depends on n. For the human genome (DNA), σ is 4 (A, C, G, and T) and n is around 3G (3 gigabases). Hence, each *Occ* entry fits in a 4-byte unsigned integer, and the footprint of *Occ* is about $4 \times 3\text{G} \times 4 \approx 48$ GB. Several techniques have been developed to reduce this large footprint based on storing only a portion of *Occ* and calculating the rest of it [33, 24, 23]. These techniques are described and analyzed in detail in the next section. In this thesis we propose a new organization of *Occ* that improves the performance of the *Count* operation, taking maximum advantage of the available memory bandwidth.

Another approach to reduce the memory usage uses wavelet trees to store the *Occ* data [46, 34]. These structures are specially space efficient when performing rank queries on large texts based on large alphabets. The efficiency of this solution also depends on the entropy of the text. A rank query on an alphabet of σ symbols is done by $log_2(\sigma)$ binary rank queries. Each binary query calculates several indexes, accesses several memory locations, and performs some arithmetic operations.

In the DNA context, the alphabet is very small and the text is relatively small and with high entropy. Therefore, optimized versions of the *Occ* matrix fit in contemporary memory systems making the space usage advantage of using wavelet trees not so relevant. However, the computational cost of wavelet trees

UNIVERSIDAD DE MÁLAGA

based algorithms is much higher than those implementations based on a table-based *Occ*.

2.2.6 Previous Works on Accelerating FM-index

FM-index implementations for specific architectures or accelerators have been published, including GPUs (Arioc [111], CUSHAW2 [74], BarraCUDA [62], [18]), Clusters (CUSHAW3 [45]), Clouds (BigBWA [3]) and FPGAs (FHAST [32]).

Several works focus on improving the performance of the exact matching algorithm (FM-index) for GPUs, like Chacon et al. [22] and Chen et al. [26]. FM-index is also included in the NVBIO [85] library, developed by Nvidia to speed up bioinformatics using GPUs and CUDA technology.

The most relevant operation in the FM-index backward search algorithm is the *rank* operation [54]. This operation, together with the *select* one, has been addressed in numerous papers which focus on optimizing both the memory footprint and the pattern search time [77]. Most of these papers are based on succinct data structures [93], [43], [42] and wavelet trees [38], [44].

2.3 Non-exact Matching using FM-index

FM-index is used in several applications in order to search for a non-exact patterns in a reference text. This is particularly useful in some fields like bioinformatics and specifically, for sequence alignment.

For sequence alignment, exact matching is insufficient. Alignments usually may contain some mismatches, due to errors in the sequencing process, differences between reference and query organisms, or both. However, there are some algorithms, built upon FM-index, able to quickly find approximate occurrences of the pattern.

There are several alignment applications relying upon FM-index variations, such as HISAT [58], Bowtie [65], Bowtie2 [64], BWA [69], [70] and SOAP [72], [76]. Bowtie and BWA are two of the most important ones and are described in the following sections.

UNIVERSIDAD DE MÁLAGA

2.3.1 Bowtie

Bowtie and Bowtie2 are open-source, ultrafast and memory-efficient alignment applications used for aligning both short and long DNA reads to large genomes. Bowtie relies upon Burrows-Wheeler transform and a modified FM-index algorithm able to quickly find non-exact alignments that satisfy a specified alignment policy. This algorithm uses backtracking to find the non-exact occurrences of the pattern: when the exact matching basic FM-index algorithm finds an empty range, the Bowtie algorithm selects a previously matched DNA base and replaces it for a different one, and tries to match the new 'modified' sequence (see Figure 2.3). This search is performed in a greedy way, namely, Bowtie alignment will not necessarily be the 'best', but it will find a valid alignment if it exists.

Bowtie indexes are optimized in order to use as less memory as possible. This way, a Bowtie index for the human genome uses around 2.2GB on disk, and has a memory footprint of just 1.3GB, being able to perform more than 25 million read alignments per CPU hour.

Compared to other sequence alignment tools, Bowtie is much faster than most of the alternative options. According to [65], Bowtie is between 216x and 691x faster than SOAP aligner and between 14.9x and 36.7x times baster than Maq aligner, depending on the read length, with very similar results.

Bowtie developers have released a new version of the software, called Bowtie2. It also includes support for longer reads and gapped alignments. Bowtie2 is described in more detail in section 5.2 and we use it as a test application in chapter 5.

2.3.2 BWA

BWA is, like Bowtie, an alignment application used mainly for aligning short DNA reads to large genomes. BWA inexact search algorithm recursively searches for the suffix array intervals of substrings of the reference text that match the searched pattern with no more than a fixed number of differences. This is achieved, essentially, using the FM-index backward search to sample distinct substrings from the reference genome. BWA uses a strategy to compress the genome index similar to Bowtie, using around 2.3GB for a 3GB genome, like the human one.

Compared with other similar alignment tools, according to [70], BWA is faster than other similar sequence alignment tools like Maq [8], SOAP2 [72] and Bowtie [65], aligning more sequences and achieving a higher confidence rate.

UNIVERSIDAD DE MÁLAGA

Figure 2.3: Bowtie: Inexact matching diagram

UNIVERSIDAD DE MÁLAGA

2.4 3D-Stacked Memory Architectures

In the last years, manufacturers have developed some commercial devices implementing stacked memory in order to improve performance and memory bandwidth of computing systems. There exist two architectures specially important in this field: High Bandwidth Memory (HBM) [4] and Hybrid Memory Cube (HMC) [81].

The first of them, HBM, has been developed by Samsung, AMD and SK Hynix. It is a high-performance RAM interface for 3D-Stacked DRAM. It has already been used in several commercial products, like the Intel Xeon Phi KNL [105] and several GPUs manufactured by AMD and Nvidia. More recently, in 2016, a new version of HBM (HBM2) specification has been released with higher memory speed and bandwidth.

On the other hand, HMC is another high performance RAM interface for TSVs (through-silicon vias)-based stacked DRAM memory, developed by the Hybrid Memory Cube Consortium, leaded by Micron. HMC has not been included in almost any commercial system, except for some Micron development boards. However, in the scientific community, it has been widely studied and used as basis for different Processing-In-Memory architectures (see section 2.6.1). It is able to perform simple memory operations inside the memory module thanks to the logic layer it includes.

2.5 Intel Xeon Phi Knights Landing

2.5.1 Knights Landing Architecture

Knights Landing (KNL) is the new architecture released for the Intel Xeon Phi x200 processors. It introduces novel features with respect to previous Xeon Phi coprocessors implementing Knights Corner architecture (KNC). Some of these new features are described below.

2.5.1.1 AVX-512 Instruction Set

KNL is the first architecture that includes partial support of the new vector instruction set AVX-512 [116]. This vector extension doubles the vector width with respect to the previous 256-bit AVX2.

UNIVERSIDAD DE MÁLAGA

The KNL architecture has a theoretical peak performance of 6 TFLOPS in single precision, which is triple than that of the previous Xeon Phi. This improvement is due to the doubling of vector processing units per core (two vs. one) and to the 50% frequency increase (1.5 GHz vs. 1 GHz) with respect to the previous KNC generation.

The Intel AVX-512 is composed of 9 subsets of instructions. However, only 4 are supported by KNL: AVX512-F, AVX512-CD, AVX512-ER and AVX512-PF.

2.5.1.2 KNL HBM Memory

The new Intel Xeon Phi KNL processors include on-package high-bandwidth memory (HBM) based on MCDRAM. KNL HBM is capable to deliver much higher bandwidth rates than DDR4 SDRAM. It provides up to 400 GB/s against the 90 GB/s provided by a 6-channel DDR4 SDRAM, as shown in Figure 2.4 which shows the results after running the STREAM benchmark [80] on a KNL system (see 3.3 for more details on KNL architecture).

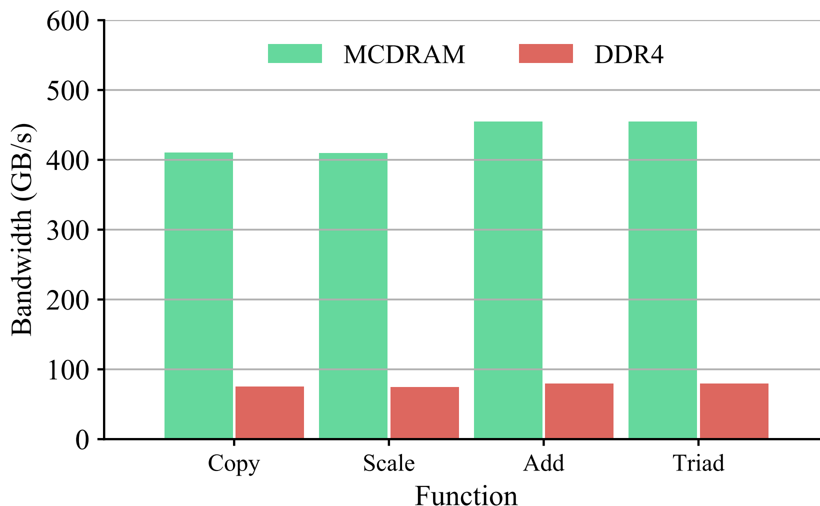


Figure 2.4: STREAM Benchmark results

HBM in KNL processors can be configured in three modes: flat, cache and hybrid [10]:

UNIVERSIDAD DE MÁLAGA

1. Flat mode: the whole HBM is used as addressable memory. The code requires modifications to take advantage of this mode, for instance, specific system calls are needed to dynamically allocate and release HBM memory.

2. Cache mode: all the HBM is used as a new level of cache. This mode requires no additional work on the code to run, but its performance may be lower than that of optimized flat mode.

3. Hybrid mode: one part of the HBM is used as cache and other as addressable memory. It has the benefits of both previous modes, but smaller available sizes for the flat and cache partitions.

In the experiments carried out in this thesis, we are using the flat mode. This configuration allows us to allocate the critical data structures in the HBM memory explicitly.

2.5.1.3 Clustering Modes

Each of the KNL cores has a private L1 cache. The L2 cache is divided into 1 MB slices that are symmetrically shared between two cores. Tiles comprising two cores and a L2 slice are connected via a 2D-mesh.

To maintain cache coherency, KNL uses a distributed tag directory organised as several tag directories which identify the state and the location of the L2 cache lines.

The KNL architecture supports three clustering modes that provide different levels of affinity between tiles, tag directories, and memory controllers [109]:

1. All-To-All: memory addresses are uniformly distributed across all tag directories.

2. Quadrant/Hemisphere: the processor tiles are divided into four or two parts, mapping all the memory addresses to local directories.

3. SNC-4/SNC-2: the chip is partitioned into four quadrants or two hemispheres as independent NUMA nodes.

The All-To-All mode is typically worse than Quadrant/ Hemisphere for almost any application. This mode is only recommended for special and specific applications. The SNC modes add some complexity to the code because the programmer must select which memory node will the data be stored in. Besides,

UNIVERSIDAD DE MÁLAGA

we performed several tests and observed that the local latency and bandwidth measures do not show a significant improvement over the Quadrant/Hemisphere modes.

Therefore, in order to improve the tag directory locality without increasing the code complexity and without deteriorating the remote access latency and bandwidth, we use the Quadrant clustering mode for our implementations, as we will see in chapter 3.

2.6 PIM Architectures

Recently, new computer architectures and techniques are appearing to try to overcome the previously mentioned memory-wall. The most promising ones are called Processing in Memory (PIM) and Near-Data Processing (NDP). Both concepts focus on reducing the amount of time and power used in memory accesses in typical processor-centric systems by placing the data closer to the computing units (together in the case of PIM), making them more data-centric.

PIM and NDP have a relevant impact on memory-intensive applications, specially those applications accomplishing some of the following requirements:

- Low arithmetic intensity, meaning that it requires low computing power for each memory access.
- Highly parallelizable, because usually is more efficient to include more small cores than increase CPU frequency rate.
- Not very dependant on deep cache hierarchies, for example, applications with random memory access patterns.

Both PIM and NDP usually rely on new technologies, specially modern chip manufacturing techniques and Through-Silicon Vias (or TSVs). The former make possible reduce both the power consumption and area of the computing units and the latter enable fast communication inside 3D-stacked memories.

Previous works show different implementations for this kind of architectures. Some researches change minimally the classic memory chips in order to include some computing units inside them, while others base their research on the Hybrid Memory Cube (HMC)[81] or High Bandwidth Memory (HBM) [4], two different 3D-stacked memory technologies. In the case of HMC, the specification even includes a lightweight logic layer able to perform simple memory operations.

UNIVERSIDAD DE MÁLAGA

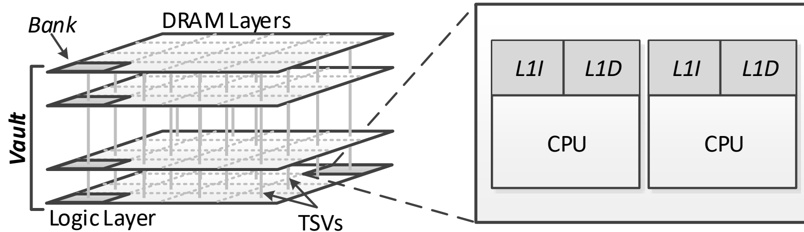


Figure 2.5: PIM Architecture diagram.

Finally, some implementations design entire new architectures. A typical PIM configuration is shown in Figure 2.5.

Concerning to their goal, there are two separated types of PIM architectures. On one hand, some approaches use general purpose processors inside the memory units. These kind of processors can be used for almost any application which requires an intensive use of memory; however, they can be power-hungry and the area used by them is relatively high. On the other hand, some architectures include specific purpose computing units, much more efficient but just useful for applications using a specific operation.

Nevertheless, PIM architectures have some relevant challenges to solve. Packing memory and computing units in the same package while keeping the power usage and temperature relatively low is one of the main challenges for these approaches. However, even more important can be the challenges derived from creating a completely new architecture which needs to be adopted by industry in the next years. This includes, for example, the development of a hardware standard being able to use PIM on real systems and a software standard to efficiently program and control PIM architectures.

2.6.1 Previous Work in PIM Architectures

A significant amount of works around this topic has appeared during the last years. Most of them, based on the Micron HMC [81] architecture, expanding or completely reworking the logical layer. For example, some works based on HMC are: [6] and [5], oriented to optimize Google PageRank and parallel graph processing, respectively; [19], analyzing the performance of Google workloads; [84] and [119], optimizing graph processing applications; [47], performing an analysis

UNIVERSIDAD DE MÁLAGA

of MapReduce workloads on HMC; [29], presenting a Near-Memory-Processing accelerator for basic data analytic operators. The work [39] includes general near-data processors in the logic layer and analyzes their performance for common applications like MapReduce, PageRank and neural networks. The papers [40] and [59], are focused on neural-network acceleration; the paper [60], with some common points with our work, improves bioinformatics applications performance through PIM; other example is [55], which analyzes density and performance of HMC and [20] that proposes a cache coherence protocol for near-data accelerators.

Furthermore, some works are oriented to use different architectures, like [50], working with NDP on GPUs, or [117], mixing CPUs and GPUs close to the data; others such as [30] introduces computational RAM and both [31] and [11], implement these techniques with commodity DRAM modules. Other example is [48] which introduces a new architecture called DIVA.

2.7 Simulators

Building real systems for testing and verification implies huge costs, completely unfeasible for research purposes. Due to that, computer architects need reliable simulation and modeling techniques in order to analyze different design options.

As described in [7], there are several ways to classify computer architecture simulators:

- **Functional vs. Timing.** Functional simulators do not model microarchitectural details, simulating just the functionality of the target. Instead, timing simulators keep track of all clock cycles on a simulated processor during a specific application execution. Other simulators, like ZSim [97], use a emulation-based approach, invoking both functional and timing models.
- **Application-Level vs Full-System.** Application-Level simulators are able to run only target applications without the operating system (OS) behind it, while Full-System simulators can run the entire target OS. ZSim is an example of Application-Level simulator, whereas gem5 [14] can be executed both in Full-System and Application-Level modes.
- **Trace-Driven vs Execution-Driven.** Depending on whether they simulate directly the target machine o they rely on trace files from real hardware.

Two of the most used computer architecture simulators in recent research are gem5 [14] and Zsim [97]. All simulations and experiments performed in chapter 4

UNIVERSIDAD DE MÁLAGA

have been conducted using Zsim and the ones performed in chapter 5 have been conducted using a modified version of gem5 (gem5-X [90]). For the memory model, we have used both Zsim and Ramulator [61].

2.7.1 Gem5

Gem5 [14] is a complete simulation infrastructure. It appeared from the union of the best aspects of M5 and GEMS simulators. M5 provides a highly configurable simulation framework, with several ISAs and diverse CPU models, while GEMS provides a detailed flexible memory system, with support for multiple cache coherence protocols.

On the CPU side, gem5 is able to simulate most commercial ISAs (ARM, x86, MIPS, Power, SPARC and ALPHA), while on the memory side, the classic mode provides a fast and easily configurable memory system and the Ruby model [96], a memory system included in gem5 which provides a flexible infrastructure for cache coherence experimentation.

One of the most relevant advantages of gem5 is the availability of two modes of execution: System-call emulation mode, avoiding the need to model devices and the operating system; and Full-sytem, able to execute both user-level and kernel level instructions and including operative system and devices.

Gem5 simulator has been greatly extended and improved over the last years, becoming one the most used hardware simulators, and a very reliable tool for hardware architects.

2.7.2 ZSim

ZSim [97] is an architectural simulator able to simulate thousand-core sytems much faster than typical simulators (around 100-1000x). This is possible thanks to the application of several simulation techniques:

- **Fast sequential simulation using dynamic binary translation:** ZSim uses instrumentation with Pin [75] to perform dynamic binary translation (DBT), eliminating the need for functional modeling of x86 and placing most of the work on the instrumentation phase.
- **Scalable and sccurate parallel simulation:** it uses parallel simulation for modeling multi-core chips, using an event-driven parallelization technique to improve accuracy.

UNIVERSIDAD DE MÁLAGA

This simulator is also focused on flexibility and usability, being able to support two main types of core models:

- **Simple core:** Small core with IPC=1 for all but load/store instructions.
- **OoO core:** Out of order modern cores with much more functionality present in real-life processors, as branch predictor, complex instruction fetching, etc.

2.7.3 Ramulator

Ramulator [61] is a fast and cycle-accurate simulation tool for current and future DRAM systems. It is able to accurately provide models for a variety of different memory standards, as, DDR3, DDR4, LPDDR, GDDR5, HBM, SALP, HMC, etc.

It can be used in two different ways:

- **Integrated:** with an architecture simulator, like gem5 or ZSim.
- **Standalone:** being fed with a memory trace or an instruction trace.

In chapter 4, we use Ramulator to model the memory system. Ramulator is more accurate and flexible than ZSim integrated memory model.

2.8 Roofline Model

The Roofline model [95, 110] is a method that provides the upper bound of performance for an application running in a specific system, generally a multi-core, many-core or accelerator processor architecture.

This model is very useful to provide insights to programmers and architects in order to improve both parallel software and hardware when working on a specific application.

With the memory bandwidth being a constraining resource, a model able to relate processor performance to memory bandwidth becomes really important. The Roofline model shows the main constraining resources for each implementation and computing hardware, helping us to change code or hardware to run desired kernels optimally.

UNIVERSIDAD DE MÁLAGA

The Roofline model is based on the concept of *operational intensity*, defined as the ratio of the number of operations (often defined as floating points operations per second) to the amount of data traffic (usually in bytes).

The result of this model is a two-dimensional graph with lines showing both the peak operational (for example, floating-point) performance (modeled as a horizontal line) of the system and the peak memory bandwidth, caches and offchip (modeled as a diagonal line starting on 0). Those two lines intersect at the point of peak computational performance and peak memory bandwidth.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

3 Split Bit-Vector Sampled FM-index Exact Matching

The emerging of data-intensive workloads that we are experiencing in recent times has led to a great interest in the design of techniques for their efficient and scalable processing. From the storage viewpoint, performance is boosted if the data is properly organized in main memory and the reference patterns exploit data access locality. However, handling a big amount of data becomes a hard challenge when unpredictable access patterns are present. The large and deep cache hierarchies found in modern processors work inefficiently with such workloads, and frequently cause a high demand of memory bandwidth.

To satisfy this high memory bandwidth requirements, industry started to apply 3D-stacked manufacturing technologies to DRAM. Such initiatives resulted in DRAM interfaces like Hybrid Memory Cube (HMC) [56], by Micron Technology, and High Bandwidth Memory (HBM) [4] from AMD and Hynix, more extensively explained in chapter 2.

The availability of these memory technologies, however, may not be enough to achieve high performance if the application presents a very low data access locality and operational intensity. In this chapter we tackle this challenge by properly organizing the application data structures, with the aim of minimizing the memory bandwidth demand. The FM-index [33] is used as a case study throughout the thesis. It allows efficient pattern searching in large reference texts. As a result, it has been applied with great success to sequence alignment [71]. However, searching algorithms based on FM-index exhibit non-uniform memory access patterns that cause frequent cache misses. In this chapter we present a new organization of the FM-index data structure capable of drastically reducing memory bandwidth requirements for the searching process. As a result, the

UNIVERSIDAD DE MÁLAGA

operational intensity of the search algorithm increases significantly, offering the opportunity to take better advantage of the available memory bandwidth.

This chapter analyzes the searching method that allows exact pattern matching based on the FM-index data structure. The study focuses on bottlenecks related to data access patterns and computational capabilities. The evaluation assumes a batch or offline setting, where a bulk of queries is issued to be processed as fast as possible.

The main contributions of this chapter can be summarized as follows:

- Searching algorithms built upon FM-index are analyzed, focusing on those aspects related to computing costs and access to data, which have a great impact on performance.
- Proposal of a new organization of the FM-index data structure layout and codification denoted *split bit-vector sampled FM-index* ($bvSFM_k$), which reduces the required traffic between memory and processor cores for the exact search process.
- An optimized search algorithm has been implemented based on the proposed FM-index data structure layout and evaluated in last generation processors such as Skylake, Broadwell and KNL. We will show that our optimized algorithm exploits the ultra high-bandwidth memory modules integrated in the KNL processor.

3.1 Analysis of FM-index variants

FM-index is a compressed data structure that allows fast searching queries [33]. The FM-index search algorithm has been presented in section 2.2.4 and is based in two operations: *Count* and *Locate*, being the latter simpler than the first one (see subsection 2.2.4.2).

This section presents a computational analysis, in terms of memory and performance, of the *Count* operation for different versions of the FM-index proposed in the literature using a table-based *Occ* structure.

The following subsections present the original (basic) FM-index [33], the overlapped version presented in [23], the FM-index with sampled *Occ* [33, 24] and the version that searches for several symbols in each iteration [23]. Finally, subsection 3.1.4.1 analyzes the size of the data structures, the memory access pattern and the potential performance of the considered versions of the algorithm. This

UNIVERSIDAD DE MÁLAGA

analysis serves as the basis to justify and develop our proposal, presented in section 3.2.

3.1.1 Basic FM-index

The *basic FM-index* assumes that Occ is a pre-computed full look-up table, that is, it stores counts for each possible symbol and index. It can be used to quickly locate the occurrences of a pattern (query) $Q[1..p]$ in a text $T[1..n]$, being $p \ll n$. An exact matching algorithm using FM-index has been presented in [33]. Figure 3.1 illustrates the *Count* step of this algorithm, called *backward search* (BS). At the end of the algorithm, the *sp* and *ep* variables contain the start and the end indices in the suffix array of T that contains Q as a prefix, respectively.

```
Algorithm BS: Backward Search Based on FM-index
Input: FM-index of T (C & Occ), Q query, n=|T|, p=|Q|
Ouput: (sp,ep): Interval pointers of Q in T
    begin
1:  sp = C[Q[p]]
2:  ep = C[Q[p]+1]
3:  for i from p-1 to 1 step -1  }
4:     sp = LF(Q[i],sp)           }
5:     ep = LF(Q[i],ep)           } 2 LFM-chains
6:  end for                       }
7:  return (sp+1,ep)
    end
```

Figure 3.1: Basic backward search algorithm based on FM-index

The main operation in the backward search algorithm is a *Last-to-First Mapping* (LFM), which is performed by calling the function $LF()$, defined as follows:

$$LF(Q[i], u) = C[Q[i]] + Occ[Q[i], u], \tag{3.1}$$

where i is the index of the loop and u is either *sp* or *ep*.

Each iteration of the loop $3-6$ in the BS algorithm accesses the Q string and makes two calls to the $LF()$ function, one with *sp* and the other with *ep*. Note that in every loop iteration, *sp* (*ep*) is updated using the value computed in the previous iteration. That constitutes two dependency chains of calls to $LF()$, one for *sp* and the other for *ep*. We denote these chains *LFM-chains* (see Figure 3.1 and Figure 3.2).

UNIVERSIDAD DE MÁLAGA

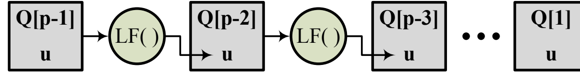


Figure 3.2: a LFM-chain, where u is either sp or ep

3.1.2 Sampled FM-index

The basic FM-index requires a large amount of memory space (see equation (2.1)) but the BS algorithm exhibits low computing cost. The *sampled FM-index* is a variant of the basic version that introduces a trade-off between memory footprint and computing cost [24, 33].

The storage requirements can be reduced by replacing the Occ structure with a smaller one that we denote $rOcc$. The structure $rOcc$ stores one column out of every d columns of Occ, that is, $rOcc[c, i] = Occ[c, 1+(i-1)\times d]$. Figure 3.3 depicts this new sampled data structure.

In order to reconstruct the content of Occ, both $rOcc$ and BWT are needed. Let us see an example for d=5. Consider that we need to know the number of occurrences of the symbol s up to the entry 8 of the BWT text, that is, the value of $Occ[s, 8]$. The nearest entry in the row s of Occ previous to $Occ[s, 8]$ that is stored in $rOcc$ corresponds to $rOcc[s, 2]$, because $\lfloor$(8-1)/d+1$\rfloor$=2. So, we can obtain $Occ[s, 8]$ by adding the value $rOcc[s, 2]$ (which is equal to $Occ[s, 6]$) and the number of occurrences of the symbol s in the sub-string of BWT from position 7 to 8. This equivalence can be expressed in general as follows:

$$\begin{aligned} Occ[s,p] &= Occ[s,q] \; + \; occur(s, BWT[(q+1)..p]) \\ &= rOcc[s, \lfloor (p-1)/d+1 \rfloor] \; + \; occur(s, BWT[(q+1)..p]), \end{aligned} \tag{3.2}$$

being q=1+$d\times\lfloor(p-1)/d\rfloor \leq p$, and *occur(s,str)* the number of occurrences of the symbol s in the string str.

A way of improving data locality consists of placing next in memory columns of $rOcc$ and the blocks of BWT required to reconstruct Occ. This is accomplished in two steps (see Figure 3.3). Firstly, rearranging the BWT text in an array of substrings of d consecutive symbols taken from BWT, called *buckets* [33]. The new data structure, named $bBWT$, is defined as $bBWT[u, v]$=$BWT[d\times(u-1)+v]$, representing the symbol v of the bucket u. Secondly, combining both $rOcc$ and $bBWT$ data structures into a new one denoted by *SFM* (*Sampled FM-index*). *SFM* associates the column j from $rOcc$ with the row j from $bBWT$. Specifically, a *SFM* row refers to a $rOcc$ column (σ counters) and the $bBWT$ bucket required

UNIVERSIDAD DE MÁLAGA

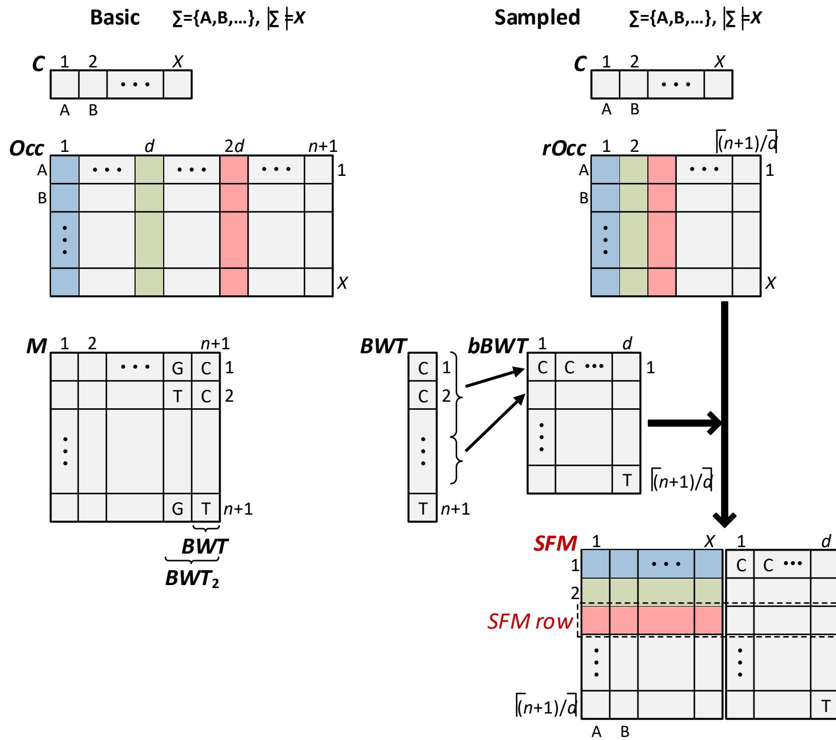


Figure 3.3: Basic and sampled FM-index data structures

to reconstruct the discarded *Occ* counters up to the next *rOcc* column.

A search algorithm based on the sampled FM-index follows a similar structure as the original backward search algorithm, but it requires to re-write the calculation of a LFM (see expression (3.1)) as follows:

$$\begin{aligned} sLF(Q[i], m, d) = C[Q[i]] + rOcc[Q[i], \lfloor (m-1)/d + 1 \rfloor] \\ + occur(Q[i], bBWT[\lfloor (m-1)/d \rfloor + 1, [1..(m \bmod d)]]). \end{aligned} \tag{3.3}$$

Note that each sampled LFM uses *rOcc* instead of *Occ*, which is d times smaller, but at the cost of performing more computation. We have to mention that the higher the value of d, the higher the computational cost.

UNIVERSIDAD DE MÁLAGA

3.1.3 K-step Sampled FM-index

The *k-step sampled FM-index* searches k symbols in a query in a single step [23]. This version reduces computing cost and improves data locality of the sampled version but increases slightly memory footprint.

To search k symbols in a single query, the original alphabet, Σ, is replaced by the set of k-tuples whose elements come from Σ (permutations with repetition). The new alphabet is denoted Σ^k and its size is σ^k. Hence, the number of counters required for C and $rOcc$ increases exponentially.

This change in the alphabet implies modifications in the sampled FM-index data structure. C is transformed into C_k, which is indexed by k-tuples in Σ^k (hence, its size is σ^k). $rOcc$ becomes $rOcc_k$, whose first dimension is also indexed by k-tuples (thus, its size is $\sigma^k \times \lceil (n+1)/d \rceil$). BWT is transformed into BWT_k, which is composed of k $(n+1)$-symbol strings, namely the last k columns of the M matrix (see section 2.2.2). These k strings, however, can be encoded as only one $(n+1)$-symbol string, where each symbol is now the concatenation of the k symbols of each row from the matrix M.

Similarly to $bBWT$, BWT_k, encoded as a single string of k-tuples of symbols, can be blocked into buckets of size d. This new structure is denoted $bBWT_k$, an array of sub-strings composed by the concatenation of d k-tuples of symbols. Just as *SFM*, the extended data structures, $rOcc_k$ and $bBWT_k$, are combined into a new one denoted by SFM_k (*k-step Sampled FM-index*). Figure 3.4 shows these new data structures for k=2. Note that Figure 3.3 represents the data structures for the one-step version (k=1).

The k-step version of the calculation of a LFM (denoted by $sLF_k()$) is an extension of the single-step version, $sLF()$ (see expression (3.3)), but using the extended Σ^k alphabet and the extended data structures: C_k, $rOcc_k$ and $bBWT_k$.

The backward search steps, $LF_k()$, are computed like the one-step version, but with an extended X^k-alphabet and with larger data structures. A single $sLF_k()$ call resolves k LFMs, that is, it is equivalent to k calls to $sLF()$. A call to $sLF_k()$ reads one piece of main memory containing the data to resolve k LFMs, exploiting in this way data locality.

Moreover, the computational cost of $sLF_k()$ is slightly higher than that of $sLF()$. However, the cost typically increases in a significantly lower rate than k factor, making the cost per LFM much lower.

UNIVERSIDAD DE MÁLAGA

3.1.4 Memory and Performance Analysis

3.1.4.1 Memory footprint

The most memory consuming data structure is Occ, for basic FM-index, and SFM_k, for the sampled variants. In the case of DNA, the full Occ matrix is a big data structure (see equation (2.1)). The sampled versions, however, reduces largely this size depending on the parameters d (sampling factor) and k (symbols searched in a single step).

Specifically, the memory footprint for the SFM_k data structure is:

$$Fp(SFM_k) = \lceil (n+1)/d \rceil \times (\sigma^k \times R + d \times \lceil log_2 \sigma^k \rceil) \text{ bits}, \quad (3.4)$$

where R is the size of a $rOcc_k$ entry.

Taking the human genome example (n=3 Gbases, σ=4) and d=64, the memory footprint for SFM_k is 1.5 GBs, for k=1, and 4.5 GBs, for k=2, considering

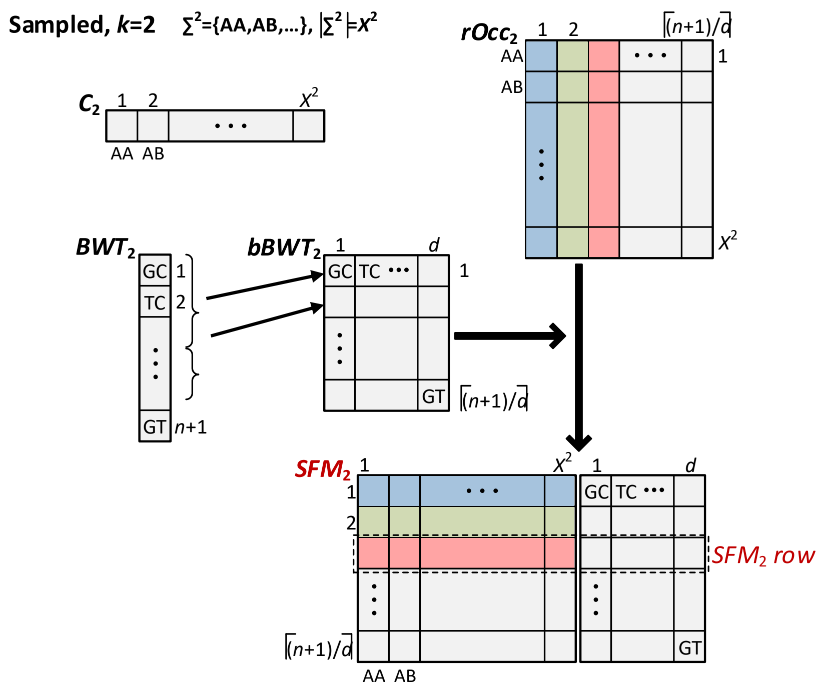


Figure 3.4: k-step FM-index (k=2)

UNIVERSIDAD DE MÁLAGA

that a *rOcc* entry fits in 32 bits. These sizes are much lower than the footprint of the full *Occ* structure (48 GBs).

Values of k greater than 2 require a large amount of memory due to the exponential dependency of $Fp(SFM_k)$ on the number of steps (k). Taking the same example as above but for k=4, the size of SFM_k increases to 51 GBs, greater than the original *Occ* structure. For this reason, in the rest of the thesis we assume that k=2.

3.1.4.2 Memory access pattern

One of the main performance limitations of the BS algorithm is related to the memory system. When executing this algorithm in an out-of-order processor, two LFM-chains are issued for each backward search query, overlapping their execution. Considering the basic FM-index, a LFM in each chain accesses both an element from the C array and other from the *Occ* matrix. The C array is very small (for DNA) and probably fits completely in the processor L1 cache. This is not the case for the *Occ* matrix, which is a very large structure. Each of these LFMs obtains the *Occ* entry address (*sp* or *ep*) using the address from the previous iteration. Given how the BS algorithm works, the *Occ* memory access pattern for a LFM-chain is not predictable and it is distributed along the whole *Occ* matrix, showing neither spatial nor temporal locality. Hence, accesses to *Occ* result in a high cache miss rate.

However, computations from both LFM-chains are partially correlated. When a part of the query has already been performed, there are usually few matches in the reference text, and the start and end pointers (*sp* and *ep*) may have similar values. In that case, the pair of LFMs executed in a loop iteration likely access two *Occ* entries that are stored in the same cache block. We have measured the ratio of these cache block correlations for query lengths of 200 and 400 symbols and different text sizes, assuming 64-byte cache blocks, and the results are depicted in Figure 3.5. From the figure, it can be noted a high degree of cache block correlation between the two $LF()$ calls within the same loop iteration.

To summarize, each pair of LFMs in a loop iteration reads two different cache blocks from main memory at the beginning of a query, but they are likely to access only one cache block when the query moves forward. Let α denote the average number of cache blocks read from main memory by each LFM pair in the same loop iteration, that is,

$$\alpha = 1 + (1 - r), \tag{3.5}$$

UNIVERSIDAD DE MÁLAGA

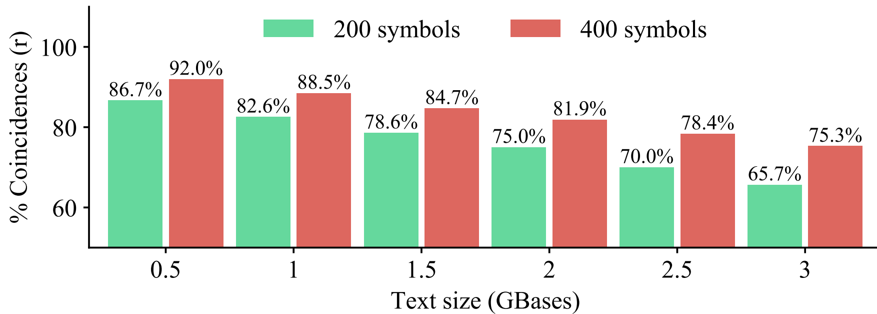


Figure 3.5: Fraction of $LF(c, ep)$ calls that access the same cache block as companion $LF(c, sp)$ calls for different text and query sizes of 200 and 400 symbols

being r the probability of sp and ep referring data from the same cache block (see Figure 3.5). The value of α depends on factors such as the index size and the query length.

As an example, taking the values from Figure 3.5, an average 200-symbol query in a 3G-base text would start with 69 miss-miss pairs (0.343×200) followed by 131 miss-hit pairs (0.657×200), resulting in a small cache hit ratio ($0.33 \approx 0.657/2$) and a $\alpha = 1.343$ ($r = 0.657$).

For the sampled FM-index variants, a LFM reads one $rOcc_k$ entry and a sub-string from $bBWT_k$ (see expression (3.3) for k=1), both belonging to the same SFM_k row. In order to minimize the number of cache blocks read from main memory, the SFM_k row has to be properly aligned to cross the minimum number of cache block boundaries. As an example, the size of a complete SFM row would be 24 bytes for $d = 32$, $X = 4$ and 32 bits (see equation (3.6)) for $rOcc$ entries, which implies that some SFM rows would cross a cache block boundary (considering 64-byte cache blocks).

$$Fp(SFM_{row}) = \sigma \times Fp(rOcc_{entry}) + d \times \lceil log_2 X \rceil \; bits. \tag{3.6}$$

Therefore, either d has a suitable value or the SFM_k row has to be padded.

Equation (3.5) has to be adapted for the sampled FM-index versions because d and k appear as new parameters. In particular, the average number of cache blocks read from main memory for each query step (performing $2k$ LFMs) is:

$$\alpha_{dk} = \Delta_{dk} \times (1 + (1 - r_{dk})), \tag{3.7}$$

UNIVERSIDAD DE MÁLAGA

where Δ_{dk} is the average number of accessed cache blocks for a pointer access (either sp or a ep), and r_{dk} is the probability that both sp and ep refer to elements stored in the same SFM_k row.

This parameter takes into account the case in which a SFM_k row occupies several cache blocks. In such case, r_{dk} is calculated assuming that for both sp and ep, either all blocks have been accessed or none of them. Δ_{dk} corrects α_{dk} for the case that only some of the cache blocks are accessed.

Table 3.1 shows the footprint of Occ and α for the basic FM-index (first row in table), as well as the footprint of SFM_k and α_{dk} for the sampled versions, using different values of k and d, and 64-byte cache blocks. The parameters r and r_{dk} were obtained experimentally searching 20M sequences of DNA strings with an average length of 200 symbols in a full human genome reference (σ=4 and n=3G). The parameter Δ_{dk} was calculated assuming random accesses to $rOcc_k$. Note that, for k=1 and d=32, the SFM_k row size is 24 bytes (see equation (3.6)), which is padded with 8 extra bytes in order to store two complete rows in a single cache block (similar situation occurs when k=2 and d=16). It is worth noting that the huge memory footprint of Occ for the basic version may result in frequent TLB misses for most of the modern processors.

Our design goal is to reduce α_{dk} for a given k value. On the one hand, Δ_{dk} increases with growing values of d, since the SFM_k row size increases. On the other hand, the greater the value of d, the higher the probability r_{dk}. For k=1 and k=2, this trade-off results in a minimum value of α_{dk} with d=192 and d=128, respectively. For 64-byte blocks, these are the sampling values that minimize the average number of cache blocks read from main memory for each query step.

Table 3.1: Basic and sampled FM-index properties (B (GB) stands for bytes (Giga bytes), CB for cache blocks and P for padding)

Basic				*Occ* column size		*Occ* size	α	SI
			r	(CB)	(B)	(GB)	(CB)	(LFMs/B)
			0.657	1	16	48	1.34	0.0232
Sampled				SFM_k row size		SFM_k size	α_{dk}	SI_{dk}
k	d	Δ_{dk}	r_{dk}	(CB)	(B)	(GB)	(CB)	(LFMs/B)
1	32	1	0.916	1	24+8P	3	1.08	0.0288
1	192	1	0.934	1	64	1	1.07	0.0293
1	448	1.57	0.943	2	128	0.857	1.66	0.0188
1	704	2.09	0.947	3	192	0.818	2.02	0.0142
2	16	2	0.860	2	72+56P	13.5	2.28	0.0274
2	128	2	0.924	2	128	3	2.15	0.0290
2	256	2.5	0.932	3	192	2.25	2.67	0.0234
2	384	3	0.936	4	256	2	3.19	0.0196

UNIVERSIDAD DE MÁLAGA

Lower d values do not exploit sp and ep locality. For higher values, the SFM_k size increment (in terms of cache blocks) outweighs the effect of pointers locality.

The low locality found in the memory access pattern and the huge size of *Occ* demand other requirements related to the memory hierarchy to achieve high performance: (i) *Occ* must be completely stored in main memory in order to avoid any page fault (e.g., at least 48 GBytes of main memory is required in the case of DNA), and (ii) the on-chip TLB hardware must be able to map all the memory required by *Occ* in order to avoid costly TLB misses.

3.1.4.3 Search intensity

The *arithmetic intensity* is the ratio of the number of operations (work) to the amount of data traffic (in bytes) [110]. In the case of the FM-index backward search, we use the number of LFMs performed per transferred byte to measure this ratio. Consequently, we name this metric *search intensity* (*SI*).

For the basic FM-index, a LFM pair needs to retrieve, in average, α cache blocks from main memory. Hence, the *SI* of the BS algorithm for a B-byte cache block is:

$$SI = \frac{2}{\alpha \times B} \text{ LFMs/byte.} \tag{3.8}$$

For the sampled versions, search intensity is calculated in a similar way, but replacing α with α_{dk} and taking into account that $2k$ LFMs are searched per query step, that is:

$$SI_{dk} = \frac{2 \times k}{\alpha_{dk} \times B} \text{ LFMs/byte.} \tag{3.9}$$

Table 3.1 shows the search intensity for the basic and sampled versions of FM-index. As it can be seen, search intensity (*SI*) is maximized for the pairs (k=1,d=192) and (k=2,d=128), obtaining similar values (0.0293 and 0.0290) and slightly better than that for the basic version (0.0232). However, SFM_k occupies three times more memory (3 GB vs. 1 GB) for the pair (k=2,d=128) than for the pair (k=1,d=192).

The impact of searching k symbols in a query step on search intensity is compensated by the increase in the average number of blocks read from memory (α_{dk}).

For example, for k=2 and d=128, the matching algorithm performs 4 LFMs in a query step (two $sLF_{k=2}()$) instead of the 2 LFMs performed with k=1 and

UNIVERSIDAD DE MÁLAGA

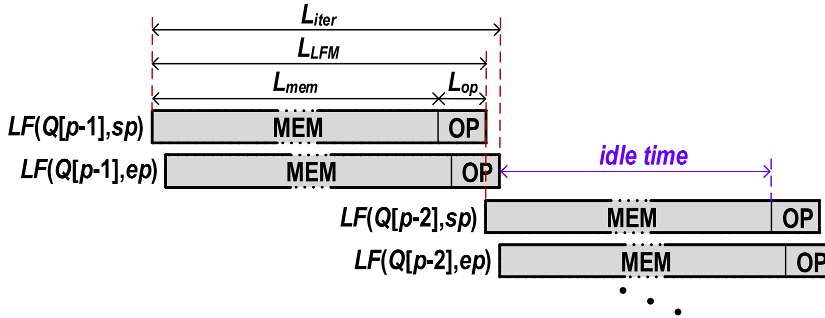


Figure 3.6: Backward search timing model, where L_x represents latencies for the different phases.

d=192. However, each step loads from memory an average of 2.15 cache blocks instead of 1.07, leaving the search intensity almost unchanged.

Taking a previous example with a 200-symbol query in a 3G-base text and 64-byte cache blocks, the obtained search intensity, *SI*, is 0.0232 LFMs/byte.

3.1.4.4 Throughput

Latency. Considering the basic FM-index, the execution of the pair of LFMs issued in a iteration of a search query can be modeled with two logical phases (see Figure 3.6). The first phase, *MEM*, corresponds to the memory operations associated to a LFM, which is mainly due to the access to *Occ*. The second phase, *OP*, corresponds to the computing operations of a LFM. Basically, this phase comprises the processing of three memory and one add instructions.

In an out-of-order processor with a non-blocking cache, the access to *Occ* with *ep* can be initiated while the cache is still servicing the access to *Occ* with *sp* in the same iteration (see Figure 3.6). However, the next access to *Occ* must wait to finish the execution of the corresponding LFM of the previous iteration (due to the data dependence in the LFM-chain). As the *MEM* phase requires typically hundreds of cycles (L_{mem}), while the *OP* phase lasts few cycles (L_{op}), the processor is idle most of the time (*idle time* in Figure 3.6).

Assuming that a single hardware thread per core is performing a complete query, the throughput is:

UNIVERSIDAD DE MÁLAGA

$$Th^{L}_{core} = \frac{2}{L_{iter}} \approx \frac{2}{L_{LFM}} \approx \frac{2}{L_{mem}} \text{ LFMs/s,} \tag{3.10}$$

where L_{iter} is the latency of the iteration (see Figure 3.6), and L_{LFM} is the latency of a complete computation of a LFM.

Bandwidth. Equation (3.10) determines an upper bound of single-thread throughput in terms of latencies. Memory bandwidth, on the other hand, also limits the maximum achievable throughput. Given the search intensity (*SI*), the throughput of a core (single thread) can be calculated as:

$$Th^{BW}_{core} = \frac{Th^{BW}_{system}}{N_{cores}} = \frac{SI \times BW_{system}}{N_{cores}} \text{ LFMs/s,} \tag{3.11}$$

being Th^{BW}_{system} the throughput of the complete system, N_{cores} the number of cores executing independent queries in parallel, and BW_{system} the main memory bandwidth for the complete system.

Sampled versions.

With the sampled FM-index, the throughput upper bounds determined by query latencies and memory bandwidth are calculated by equations (3.10) and (3.11), respectively, but replacing SI with SI_{dk}.

Regarding Th^{L}_{core}, the LFM latency (L_{LFM}) increases compared with the basic version because of the larger instruction count in the computing phase (OP). Hence, the maximum throughput per core is lower in the sampled versions.

Regarding Th^{BW}_{core}, throughput bound is maximum for k and d values that maximize SI_{dk} (see Table 3.1).

3.1.5 Optimizing Throughput: Overlapped FM-index

The basic and sampled FM-index variants have different characteristics in terms of memory footprint and data locality exploitation. However, their impact on throughput is much less important as the search intensity remains almost unchanged (see Table 3.1).

The exact matching algorithm (BS algorithm) is typically query latency bound, since many cycles are lost waiting for data (*idle time* in Figure 3.6), and as a

UNIVERSIDAD DE MÁLAGA

```
Algorithm OBS: Query-Overlapped Backward Search
Input: FM-index of T text (C & Occ), Q[] array of queries
Input: n:|T[]|, N_q:|Q[]|, p:|Q[k]{}|, k=1...N_q
Ouput: (sp[k],ep[k]): Interval array of pointers of Q[k] in T
     begin
1:   sp[k] = C[Q[k]{p}], k=1...N_q
2:   ep[k] = C[Q[k]{p}+1], k=1...N_q
3:   for i from p-1 to 1 step -1
4:     for k from 1 to N_q step 1
5:       sp[k] = LF(Q[k]{i},sp[k])
6:       ep[k] = LF(Q[k]{i},ep[k])
7:       prefetch(Occ[Q[k]{i},sp[k]])
8:       prefecth(Occ[Q[k]{i},ep[k]])
9:     end for
10:  end for
11:  return (sp[k]+1,ep[k]), k=1...N_q
     end
```


Figure 3.7: Backward match algorithm overlapping N_q queries

consequence, wasting part of the available memory bandwidth. However, the memory latency responsible of the *idle time* can be hidden by issuing a given number of different independent queries, that is, by overlapping the memory accesses of several queries (batch or offline processing). This way, the throughput upper limit imposed by query latencies is increased. The high number of queries which are usually involved in solving genome mapping problems makes this approach feasible.

The resulting algorithm that we have denoted as Overlapped Backward Search (OBS) is shown in Figure 3.7 for the basic FM-index. The OBS algorithm executes a total of $2N_q$ LFM-chains for each iteration of the outer loop 3−9, corresponding to an array of N_q different queries that are searched concurrently. Note that after computing the two LFMs required for a given query, two prefetch operations are issued to retrieve from memory the two *Occ* entries needed for computing the next two LFMs of the same query. The latencies of these memory reads are hidden by computing LFMs from other queries (see Figure 3.8). By overlapping independent queries, the processor should be busy most of the time.

Assuming a single hardware thread per core, the minimum number of LFMs that must be overlapped in order to nullify the idle time is L_{iter}/L'_{op}, where L'_{op} is the latency of the fraction of the OP phase that is not overlapped with the same phase of other LFMs (see Figure 3.8). Therefore, in order to reach such situation, N_q must take a value such that $2 \times N_q = L_{iter}/L'_{op}$. We calculate L'_{op} for several implementations of the algorithm and for several processors in section 3.3.

UNIVERSIDAD DE MÁLAGA

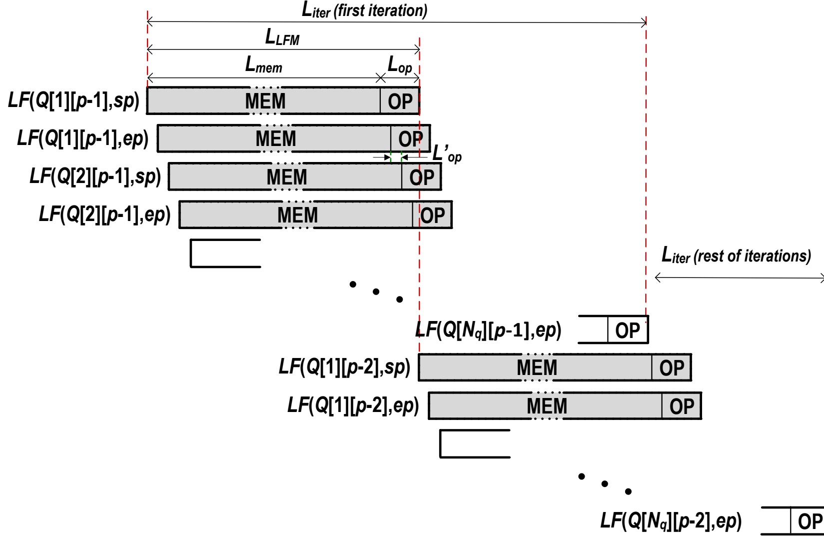


Figure 3.8: Backward search timing model with N_q overlapped queries

Using the above expression, the maximum throughput obtained by a core with the OBS algorithm is:

$$Th^{C}_{core} = \frac{2 \times N_q}{L_{iter}} = \frac{1}{L'_{op}} \text{ LFMs/s.} \tag{3.12}$$

Current architectures support simultaneous multithreading (SMT) [108], a technique that allows a single core to execute several interleaved independent execution flows (hardware threads). In this situation, the N_q queries can be distributed among the hardware threads of a core.

3.2 Split Bit-Vector Sampled FM-index

The sampled variants of FM-index have allowed to reduce memory footprint and to improve data locality compared to the basic version. However, the impact on search intensity is limited, as the increase in the number of LFMs per query is somehow compensated by the increase in α_{dk}, the average number of cache blocks accessed (see equation (3.7)). For example, as shown in Table 3.1, for k=2 and d=128, the sampled matching algorithm solves two symbols (four LFMs) in a single query, but it requires to load two cache blocks instead of one (basic

UNIVERSIDAD DE MÁLAGA

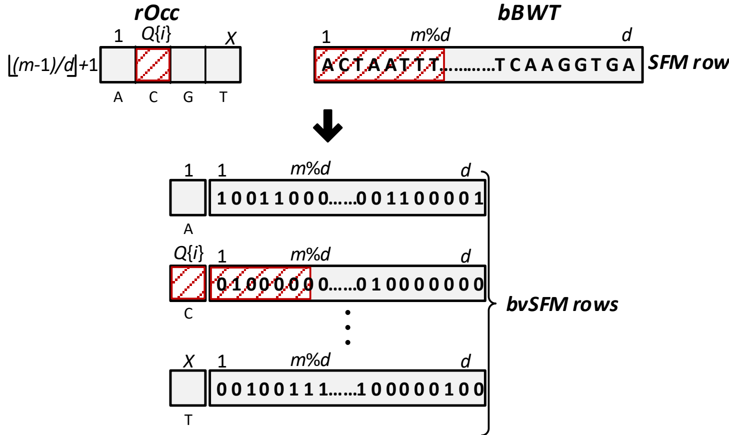


Figure 3.9: FM-index data structures
Original SFM_k (k=1) data structure (top) and new $bvSFM_k$ data structure (bottom). The accessed data during the computation of a LFM is marked in red (see expression (3.3)).

version), leaving the search intensity almost unchanged. According to equation (3.11), the throughput limited by memory bandwidth is not improved.

In order to increase search intensity to improve bandwidth throughput, the parameter α_{dk} must be reduced. The upper part of Figure 3.9 shows a row of the SFM_k data structure, for k=1. All entries accessed in the computation of a LFM are marked in red, according to expression (3.3). Note that only a single entry in $rOcc$ is accessed, so that if σ is large enough the substring accessed in $bBWT$ may be stored in a different cache block. This occurs, for example, in the case of DNA (σ=4) and k=2. Since the alphabet size is 16 (σ^k), and each entry in $rOcc$ has a size of 4 bytes, a single $rOcc_2$ column occupies a complete 64-byte cache block (16 counters × 4 bytes/counter), forcing the whole SFM_k entry to occupy several cache blocks, as shown in Figure 3.10.

An approach to save cache space consists in reducing the size of the $rOcc$ entries. Ferragina et al. [33] propose a solution using a two-level structure based on buckets and super-buckets. However, for big texts, the super-bucket data would be too big to be stored on cache and would increase significantly the amount of main memory traffic.

We propose to change the SFM_k layout and data codification so that all data needed to compute a LFM is stored in a minimum number of cache blocks.

UNIVERSIDAD DE MÁLAGA

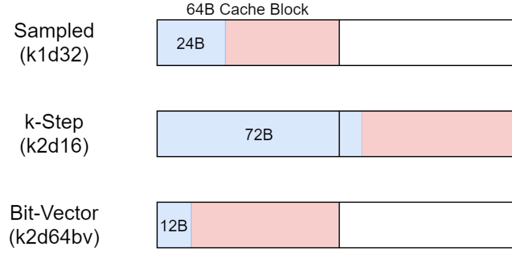


Figure 3.10: Sampled, k-Step and Bit-vector FM-Index cache mapping.

3.2.1 Our approach: Split Bit-Vector Sampled FM-index

We denote the new data structure by $bvSFM_k$, called *split bit-vector sampled FM-index*. Our solution comes from the observation that only one out of the σ^k $rOcc_k$ entries is read for each LFM computation (see upper part of Figure 3.9).

The $bvSFM_k$ structure is obtained from SFM_k through two transformations:

1. Partition each row of SFM_k into σ^k rows, where each of them consists of a single $rOcc_k$ entry combined with the complete associated bucket. Specifically, the row t of SFM_k, that is, $SFM_k[t,*] \equiv rOcc_k[*,t] \mid bBWT_k[t]$ (a concatenation of the column t of $rOcc_k$ and the bucket t), is transformed into σ^k rows of $bvSFM_k$, of the form, $bvSFM_k[(t\text{-}1)\sigma^k{+}i,*] \equiv rOcc_k[i,t] \mid bBWT_k[t]$, for i=1,...,σ^k.

2. Compression of each bucket using a bitmap where each symbol is represented by a single bit. This representation is as follows: given the row $bvSFM_k[(t\text{-}1)\sigma^k{+}i,*]$ $(1{\leq}\, i \leq \sigma^k)$, the corresponding bucket ($bBWT_k[t]$) is replaced by a bitmap of length d, where a symbol in the bucket is represented by a set bit (1) if it is equal to the one associated to the entry $rOcc_k[i,t]$, and by an unset bit (0) otherwise.

The lower part of Figure 3.9 shows, as an example, the σ^k rows of $bvSFM_k$ for k=1. Note that now, all data required to calculate a LFM (in red) are placed together in memory and in a compact way (As shown at the bottom part of Figure 3.10), minimizing memory bandwidth consumption. The transformation also allows the *occur()* function in expression (3.3) to be simplified. Now, it has to count the number of set bits (1) in the accessed bucket, operation that can be efficiently performed by the *popcount* instruction.

UNIVERSIDAD DE MÁLAGA

Table 3.2: Split bit-vector k-step sampled FM-index parameters, for $k = 2$

d	$bvSFM_k$ (CB)	row size (B)	$bvSFM_k$ size (GB)	α_{sk} (CB)	SI_{sk} (LFMs/B)
32	1	8	12	1.108	0.0564
64	1	12+4P	12	1.088	0.0574
96	1	16	8	1.081	0.0578
224	1	32	6.86	1.069	0.0585
480	1	64	6.4	1.061	0.0589

3.2.2 Memory footprint of our approach

The new $bvSFM_k$ data structure has σ^k rows for each row of the original SFM_k structure, as shown in Figure 3.9. Therefore, the memory footprint of $bvSFM_k$ is:

$$Fp(bvSFM_k) = \sigma^k \times \lceil (n+1)/d \rceil \times (R+d) \text{ bits}, \tag{3.13}$$

where R is the size of the $rOcc_k$ entry. Note that the row size does not depend on the alphabet size (σ^k).

Table 3.2 shows the size of a $bvSFM_k$ row and of the complete structure for the human genome for k=2 and different values of d. For all the selected d values, a $bvSFM_k$ row fits in a cache block. For instance, with d=64, the $bvSFM_2$ row size would be 12 bytes, assuming 32-bit $rOcc_k$ counters.

Compared to the corresponding SFM_k values , the size of the whole data structure increases. For instance, with d=64, the $bvSFM_{k=2}$ and $SFM_{k=2}$ footprints are 12 GB and 4.5 GB, respectively. Current computing systems have enough memory to allocate this up-sized $bvSFM_{k=2}$ data structures.

Nevertheless, only high-end servers can allocate $bvSFM_{k=3}$ footprints, and for example, none of k=3 versions fits in the up to 16 GiB MCDRAM integrated on Intel KNL processors.

3.2.3 Search Intensity and Throughput

To minimize memory traffic, a value must be chosen for d such that a $bvSFM_k$ row fits in a single cache block. In addition, these rows must be cache-aligned in order to avoid splitting a row between two consecutive cache blocks. That means that the cache block size must be an integer multiple of the row size. Otherwise, the rows must be padded accordingly.

Table 3.2 shows α_{dk} and SI_{dk} for different values of the sample distance d,

UNIVERSIDAD DE MÁLAGA

assuming 64-byte cache blocks and k=2. Search intensity is calculated using equation (3.9). These values have been obtained in the same way as those of Table 3.1. Different from the $SFM_{k=2}$ rows, which require two or more cache blocks to be stored, the rows of $bvSFM_{k=2}$ fit in a single cache block. As a result, the value of α_{dk} is lower for the bit-vector version, and consequently its search intensity is about twice larger than that of SFM_k. In addition, the sensitivity of SI_{dk} with d is minimum for *bvSFM*, as its row size does not depend on d for the selected values.

The throughput upper bounds are calculated by equations (3.10) and (3.11). The proposed split bit-vector version improves SI_{dk} of the backward search algorithm and, hence, increases the throughput upper bound given by memory bandwidth.

In addition, as with the basic and sampled versions, it is feasible to combine this data structure with the overlapping of independent queries (OBS algorithm). Therefore, the backward search algorithm on $bvSFM_k$ that we propose is similar to the OBS algorithm but implementing the count of set bits (1) when the *occur()* function is called during a LFM computation, as explained in section 3.2. The resulting search algorithm also improves the throughput limited by query latencies (see equation (3.12)).

3.3 Throughput Bounds Analysis

In this section, we assess the throughput bounds of the backward search algorithms based on the analyzed versions of FM-index. In particular, we consider the sampled versions with pairs:

- (k=1,d=32), denoted as k1d32-SFM,
- (k=1,d=192), denoted as k1d192-SFM,
- (k=2,d=16), denoted as k2d16-SFM,
- (k=2,d=128) denoted as k2d128-SFM,

and the split bit-vector version with pairs:

- (k=2,d=64), denoted as k2d64-bvSFM,
- (k=2,d=96), denoted as k2d96-bvSFM.

UNIVERSIDAD DE MÁLAGA

Table 3.3: Features of processors used in the evaluation

	Xeon E5-2630V4 (Broadwell)	Xeon Gold 5120 (Skylake)	Xeon Phi 7210 (KNL)
Cores	10× @ 2.2 GHz	14× @ 2.2 GHz	64× @ 1.3 GHz
IPC	4	4	2
HW Threads	2	2	4
Vector Unit	AVX2	AVX-512	AVX-512
Memory Peak BW	- 68 GB/s (DDR4)	- 107 GB/s (DDR4)	400 GB/s (MCDRAM) 95 GB/s (DDR4)
Memory Size	- 256 GiB (DDR4)	- 48 GiB (DDR4)	16 GiB (MCDRAM) 192 GiB (DDR4)

Table 3.4: Instruction count and computation latency per $sLF_k()$ call

		Broadwell		Skylake		KNL	
	Instr.	L'_{op}		L'_{op}		L'_{op}	
Algorithm	count	(cycles)	(ns)	(cycles)	(ns)	(cycles)	(ns)
k1d32-SFM	33	8.25	3.8	8.25	3.8	16.5	12.7
k1d192-SFM	77.5	19.38	8.8	19.18	8.8	38.75	29.8
k2d16-SFM	37.5	9.38	4.3	9.38	4.3	18.75	14.4
k2d128-SFM	98.5	24.62	11.2	24.63	11.2	49.25	37.9
k2d64-bvSFM	23.5	5.88	2.7	5.88	2.7	11.75	9.0
k2d96-bvSFM	38	9.5	4.3	9.5	4.3	19	14.6

All versions use the query-overlapped technique (OBS) to maximize throughput. (k,d) values have been selected so that a SFM_k row occupies the minimum number of cache blocks. k1d32-SFM, k2d16-SFM and k2d64-bvSFM have 64-bit buckets, which matches the maximum data size of a 64-bit processor. We have not studied the basic sequential algorithm (without task-parallel) because of its very low throughput.

The assessment is carried out in three processor architectures from Intel whose relevant features are given in Table 3.3.

3.3.1 Instruction count

Table 3.4 shows the average number of instructions in the *OP* phase (see Figure 3.6) of the calculation of a LFM for the different versions of the backward search algorithm, as well as, the time spent by the processor to execute those instructions. We have analyzed optimized x86 machine codes. Note that both the Broadwell and the Skylake processors work at 2.2 GHz and execute 4 instructions per cycle, while the KNL processor works at 1.3 GHz and executes 2 instructions

UNIVERSIDAD DE MÁLAGA

Table 3.5: Throughput limits imposed by query latencies and memory bandwidth (in LFMs/s) for Broadwell and Skylake architectures. Underlined entries show the minimum throughput

FM-index version	Broadwell				Skylake			
	Computing		Bandwidth		Computing		Bandwidth	
	Core	System	Core	System	Core	System	Core	System
Basic	36M	360M	105M	1.05G	36M	500M	117M	1.64G
Overl. FM	568M	5.68G	105M	1.05G	568M	7.95G	117M	1.64G
k1d32	267M	2.67G	126M	1.26G	267M	3.73G	140M	1.96G
k1d192	114M	1.14G	128M	1.28G	114M	1.59G	143M	2G
k2d16	469M	4.69G	128M	1.28G	469M	6.57G	133M	1.87G
k2d128	179M	1.79G	135M	1.35G	179M	2.50G	141M	1.98G
k2d64-bv	749M	7.49G	251M	2.51G	749M	10.49G	279M	3.91G
k2d96-bv	463M	4.63G	251M	2.52G	463M	6.48G	281M	3.94G

per cycle.

The Intel IACA tool [53] was used to analyze the hardware resource occupation in Broadwell and Skylake processors, while a similar analysis was made manually in KNL because the IACA tool does not support this architecture. In all processors, the resource that limits most the computation of the LFMs is the front-end unit, in charge of processing 2 (KNL) or 4 (Broadwell, Skylake) instructions per cycle.

As expected, k1d32-SFM, k2d16-SFM and k2d64-bvSFM versions (those with 64-bit buckets) have low instruction counts because the operations in *occur(s,str)* (see expression (3.2)) translate into a few processor instructions if the bucket size is equal or smaller than the processor word (64 bits). Likewise, those versions with larger d values have higher instruction counts because they have to loop through d-symbol buckets.

3.3.2 Throughput Bounds

Tables 3.5 and 3.6 show the throughput upper bounds determined by query latencies (processor computing capacity) and main memory bandwidth for all FM-index versions. These values have been obtained through the equations (3.12) for computing time and (3.11) for memory bandwidth, taking as BW_{system} those values shown in Figure A.3 from Appendix A.

UNIVERSIDAD DE MÁLAGA

Table 3.6: Throughput limits imposed by query latencies and memory bandwidth (in LFMs/s) for the KNL architecture. Underlined entries show the minimum throughput

FM-index version	KNL			
	Computing		Bandwidth	
	Core	System	Core	System
Basic	10M	640M	57M	3.64G
Overl. FM	168M	10.74G	57M	3.64G
k1d32-SFM	79M	5.04G	99M	6.33G
k1d192-SFM	34M	2.15G	101M	6.44G
k2d16-SFM	139M	8.87G	94M	6.02G
k2d128-SFM	53M	3.38G	100M	6.38G
k2d64-bvSFM	221M	14.16G	197M	12.61G
k2d96-bvSFM	137M	8.76G	198M	12.70G

As expected, the system throughput bounds imposed by memory bandwidth are much higher in KNL than in Broadwell or Skylake. For instance, for k2d128-SFM, the throughput bound in KNL is 6.38G LFMs/s while it is 1.35G LFMs/s in Broadwell and 1.98G LFMs/s in Skylake. The reason is that, in KNL, the FM-index structure is stored in its MCDRAM banks which provides a much higher bandwidth than DDR4 DRAM. The basic version in KNL has, however, a lower limit (3.64G LFMs/s) because the FM-index structure does not fit completely in the MCDRAM memory.

KNL cores have a lower computing capacity than Broadwell/Skylake cores (1.3GHz and 2 instructions per cycle versus 2.2GHz and 4 instructions per cycle) but a much higher memory bandwidth. This fact results in more versions of the backward search algorithm being compute bound for KNL than for Broadwell/Skylake (in particular, basic FM, k1-32SFM, k1-192SFM, k2-128SFM and k2-96bvSFM versions).

The split bit-vector version performs best, mainly due to its higher search intensity compared to the other versions. Specifically, the memory bandwidth throughput bound for *bvSFM* is around twice as large as that of the *SFM* versions.

In summary, the best result for Broadwell is 2.52G LFM/s and it is achieved with the k2d96-bvSFM version, with a memory footprint of 8 GB. For Skylake, the best result is 3.94G LFMs/s, obtained with the same version. Finally, for KNL, this version is compute bound, achieving the best throughput with the k2d64-bvSFM version (12.61G LFM/s), with a memory footprint of 12 GB. In

UNIVERSIDAD DE MÁLAGA

general, the split bit-vector data structure strongly improves the throughput for all processor architectures.

3.4 Experimental Evaluation

3.4.1 Experimental Setup and Methodology

The experimental evaluation was conducted on the three platforms already presented and whose specifications are shown in Table 3.3. More concretely the evaluation was conducted on:

- A system with an Intel Xeon Phi 7210 processor (KNL), 16 GiB of MCDRAM and 192 GiB of DDR4 running Ubuntu 16.04.1 Linux.
- A system with an Intel Xeon Gold 5120 processor (Skylake) and 48 GiB of DDR4 running CentOS Linux 7.
- And, a system with an Intel Xeon E5-2630v4 (Broadwell) and 256 GiB of DDR4 running Ubuntu 16.04.

We used the Intel C Compiler (ICC version 17.0.4) with common flags, `-O3 -qno-opt-prefetch`, and architecture dependent flags, `-xMIC-AVX512`, `-xCORE-AVX512`, and `-xCORE-AVX2`, for the KNL, Skylake and Broadwell systems, respectively.

Thread-level parallelism has been exploited by using all the available threads in all the physical cores. For the KNL system, all FM-index data structures were placed in the MCDRAM, and it was configured in memory flat mode and in quadrant clustering mode (see section 2.5). In addition, we used 1 GiB huge TLB pages to avoid TLB misses. The overlapping factor, N_q, was set to 4 for KNL, and to 20 for Broadwell and Skylake, being the minimum number of queries to be overlapped to keep busy the processor for all the versions.

A set of 20M queries generated by the Mason simulation tool [49] was used as input data. These queries (200 symbols in average) have been searched in the human genome reference GRCh38 (3 gigabases). All experiments were conducted after loading the sequences into memory.

UNIVERSIDAD DE MÁLAGA

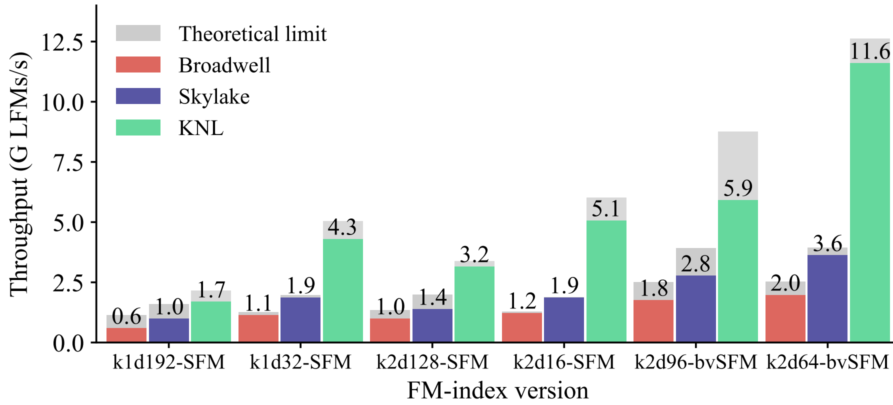


Figure 3.11: Throughput for different FM-index versions

3.4.2 Throughput

Figure 3.11 shows the throughput achieved by different FM-index versions. Each bar is split into two parts. The colored part (bottom of the bar) shows the value obtained experimentally (and shown in the figure) in each processor. The gray part (top of the bar) shows the theoretical value achievable according to the models presented in sections 3.1.4 and 3.3.

In general, the experimental values reasonably approximate the theoretical ones. The differences are due to processor features not taken into account in the models, specially, the penalty caused by branch miss-predictions. The actual throughput is about 95% of the theoretical limit for the backward search algorithms with 64-bit buckets, and it drops to around 80% for larger bucket sizes. In these cases, the count of coincidences in a variable number of 64-bit pieces forces to execute more branches, which also have unpredictable behaviour because they are dependent on the input data.

Table 3.7 shows the branch predictor statistics when executing a query of 100,000 sequences with 200 symbols each on a text with 3G symbols (those executions were carried out on several processors giving similar results).

The branch miss-prediction penalty of those versions with buckets larger than 64 bits adds to the substantial increase in the instruction count (see Table 3.4). As a result, throughput for versions with large buckets (more than 64 bits) is

UNIVERSIDAD DE MÁLAGA

Table 3.7: Branch predictor misses

FM-index version	Total	Misses	% misses
k1-32SFM	80.1M	8.6M	10.77
k1-192SFM	215.9M	36.6M	16.96
k2-16SFM	71.4M	6.6M	9.23
k2-128SFM	158.7M	25.5M	16.05
k2-64bvSFM	13.4M	0.1M	0.41
k2-96bvSFM	52.2M	8.9M	17.05

lower than that of versions with shorter buckets.

Summarizing, from Figure 3.11 we can come to the conclusion that the backward search algorithm based on the proposed split bit-vector data structure outperforms by about 60% and 90% the best of previous implementations, executed in Broadwell and Skylake processors, respectively. In KNL, our proposal outperforms by about 135% the best of previous solutions adapted to this processor. In addition, the best throughput in KNL, obtained for k2d64-bvSFM version, is about 6x and 3x that achieved by Broadwell and Skylake, respectively. This improvement is mainly due to the ultra high-bandwidth provided by the MCDRAM memory.

3.4.3 Roofline Model

The roofline model [110, 28] is a simple and intuitive visual method that provides performance upper bounds for an application running in a given architecture.

This model is based on the concept of *arithmetic intensity*. However, since the backward search algorithm does not perform floating-point operations, the *search intensity* is used instead.

Figs. 3.12, 3.13 and 3.14 show the roofline model of different FM-index backward search algorithms on the three processor architectures. The model considers the main memory peak bandwidth and the experimental results obtained when performing random memory accesses (for this we run the RANDOM benchmark described in appendix A) as the memory bandwidth bounds. This random access bandwidth is, in fact, the hardware resource that really limits the algorithm performance for the best FM-index implementation (k2d64-bvSFM) in all processor architectures. This algorithm version is able to use up to 95% of the peak bandwidth for the KNL processor (with vector extensions, AVX-512, extensively used as intrinsics in the computation of a LFM) and almost all the available bandwidth for the Broadwell and Skylake processors.

UNIVERSIDAD DE MÁLAGA

Table 3.8: Match operation performance

Implementation	Performance (GLFM/s)	Index Size (GB)
sdsl-lite library on Broadwell	0.122	1.25
sdsl-lite library on Skylake	0.147	1.25
sdsl-lite library on KNL	0.455	1.25
2-Step + AC on Intel Xeon E5-2650 CPU [22]	0.5	3
2-Step + AC on NVIDIA Kepler GTX Titan GPU [22]	3.8	3
NVBIO on Tesla P100*	2.7*	0.23*
*Test performed using a reduced 950 MiB reference file		

3.4.4 Comparison with Other Implementations

Table 3.8 shows the performance of different FM-index implementations presented in the literature when executing the input data queries described in 3.4.1 (except when stated otherwise):

- **sdsl-lite** [42, 99] is a powerful and flexible library which implements several succinct data structures. Succinct data structures focus on representing an object in space close to the information-theoretic lower bound while supporting operations of the original object efficiently. The results reported in table 3.8 use Huffman shaped wavelet trees with no compression.

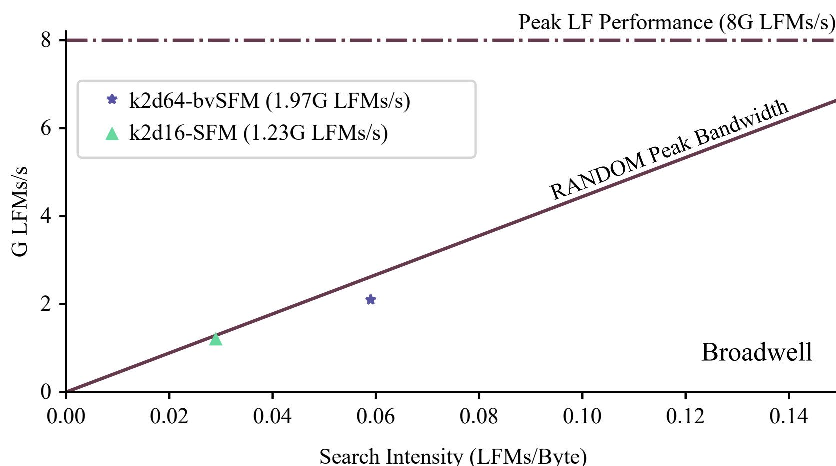


Figure 3.12: Broadwell roofline model

UNIVERSIDAD DE MÁLAGA

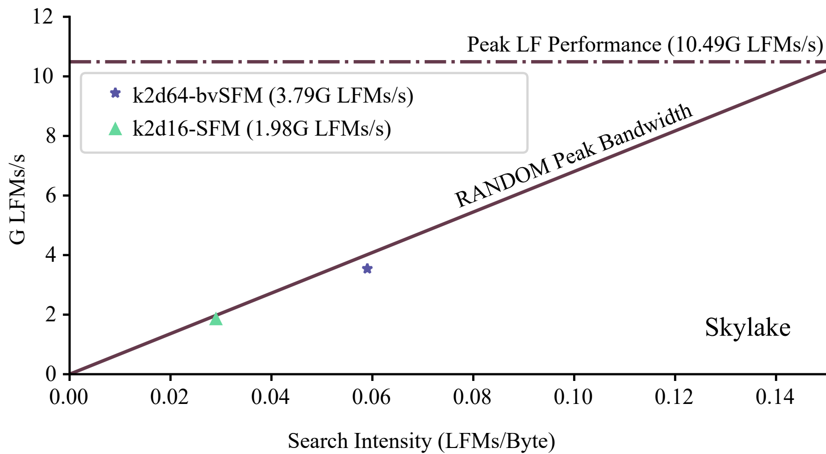


Figure 3.13: Skylake roofline model

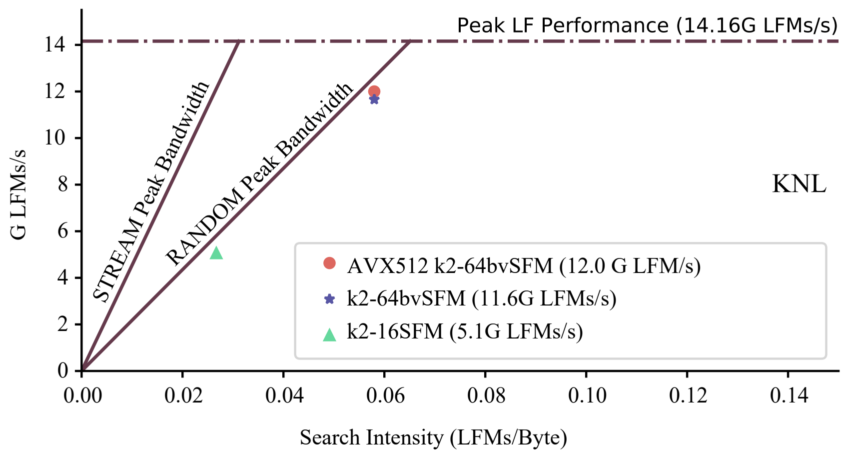


Figure 3.14: KNL roofline model

- **2-steps with alternate counters** (AC) performance shown in table 3.8 is reported in [22]. They use the 2-steps FM-index version already analyzed in this chapter and another variation called alternate counters. This variation reduces the amount of counters to store, and thus, the whole footprint of the data structure. However, this has two major disadvantages: i) increase in

UNIVERSIDAD DE MÁLAGA

the computing power required; and ii) in a CPU, the amount of cache blocks to bring from memory to caches increase significantly, reducing throughput.

- **NVBIO** [85] is a GPU-accelerated framework for sequence alignment. It is a modular library developed by NVIDIA which uses CUDA to improve the throughput of some bioinformatic workloads. We have run some experiments using NVBIO on a NVIDIA Tesla Pascal P100, a modern GPU that includes HBM2 high-bandwidth memory technology (results are shown in table 3.8).

3.5 Related work

Many sequence alignment applications based on FM-index have emerged recently, such as HISAT [58], Bowtie [65], BWA [69], [70] and SOAP [72], [76].

Several authors focus on improving the performance of the backward search algorithm (FM-index) for GPUs, like Chacon et al. [22] and Chen et al. [26]. FM-index is also included in the NVBIO [85] library, developed by Nvidia to speed up bioinformatics using GPUs and CUDA technology.

The most relevant operation in the FM-index backward search algorithm is the *rank* operation [54].This operation, together with the *select* one, has been addressed in numerous papers which focus on optimizing both the memory footprint and the pattern search time [77]. Most of these papers are based on succinct data structures [93], [43], [42] and wavelet trees [38], [44].

Unlike mentioned previous works, our work focuses on improving the pattern search time for genomic data on CPU, specially those with many cores and high bandwidth memory. Unlike CPUs, GPUs exploit fine-grained massive parallelism, but the performance drops when the control flow diverges or the data access pattern is irregular. However, while improving pattern search time, we increase memory footprint, getting close to other fast exact matching algorithms, like suffix array binary search [78].

Even if these techniques can also be very fast, FM-index can be used for non-exact matching. Several aligment tools are based on it and could be benefited from the ideas and techniques described in this chapter.

UNIVERSIDAD DE MÁLAGA

3.6 Conclusions

This chapter presents a new data layout organization of FM-index that boosts throughput thanks to an increase in the search intensity. Basically, our optimized data structure packs all relevant data needed in a query step within a single cache block, minimizing the memory bandwidth demand.

We have experimentally evaluated the backward search algorithm based on our proposed FM-index structure using three multi-core processors. Our proposal outperforms by about 60%, 90% and 135% the best of previous implementations.

The best performance was obtained in the Intel Xeon Phi (KNL) architecture, mainly because of the high peak random access memory bandwidth. Our implementation is able to obtain a throughput of 12G LFM/s, being about 3x faster than previous GPU implementations and about 4.4x faster than the GPU version implemented in the NVIDIA NVBIO bioinformatics library executed on a Tesla Pascal P100.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

4 FM-index Exact Matching Using Processing in Memory

In previous chapter we have analyzed the behaviour of a memory intensive application, genome sequence alignment using FM-index, which exhibits irregular memory accesses. We have studied the behaviour and impact of both memory bandwidth and memory latency on this kind of application in different processor architectures. However, the issue of power consumption has not been addressed yet.

Power consumption is a major problem for modern computing centers, both for the energy consumed by the system itself, and for the intensive cooling solutions most modern computing systems require. Memory systems consumption is also very relevant for data intensive applications. This lead us to an interesting opportunity to explore new memory architectures able to reduce both the energy used per accessed byte and the total time the system is busy processing a specific workload. Some new trending memory technologies, as *Near-Data Processing* (NPD) and *Processing-In-Memory* (PIM), promise to address both problems. They try to effectively improve the energy efficiency of HPC memory systems.

As mentioned in chapter 3, applications with random and unpredictable memory accesses do not perform very well on traditional computing architectures, with deep cache hierarchies of several levels.

Most of these applications does not take any advantage from cache systems and hardware prefetching. Sometimes, they are even penalised by the latency

UNIVERSIDAD DE MÁLAGA

increase introduced by the latter. This is mainly because those applications only process little data from a cache block before accessing another block from remote memory areas.

The recent proliferation of those workloads in several fields of HPC brings us to the development and analysis of new computer architectures. Our work focuses on improving the throughput and energy efficiency of low-computing (arithmetic intensity), memory intensive applications presenting random memory access, with special focus on our FM-index based application.

As it has been shown in chapter 3, FM-index is an example of this kind of application. It exhibits a completely random and unpredictable access pattern and requires low computing power per each accessed memory block, making it a good candidate to be implemented on NDP/PIM architectures.

Our main contributions presented in this chapter can be summarized as follows:

- We propose three system architectures for evaluating random access applications. Those architectures are divided in two conventional DDR setups and a PIM setup.
- We analyze the throughput of random access applications on several simulated systems, comparing setups when the processing is performed near to the data with typical computing systems.
- A study of the energy efficiency of all the different analyzed systems has been conducted.
- The same evaluation has been carried out with the FM-index based exact matching algorithm used as a case application.

For the evaluation of our proposals, we use a well-known hardware simulator called ZSim [97] and an integrated power, area and timing modeling framework called McPAT [73].

4.1 Hardware Design and Simulation

In this section we present the characteristics of the architectures we propose to evaluate the performance of random access applications. Two of them are setups based on DDR memory systems and one is a PIM architecture.

UNIVERSIDAD DE MÁLAGA

Table 4.1: Hardware simulation setups summary

	DDR Setup 1	**DDR Setup 2**	**PIM**	**Intel i7-8700**
Cores	64 @ 2.4 GHz	36 @ 3.6 GHz	64 @ 1.5 GHz	6 @ 3.2-4.6GHz
Core type	OoO	OoO	in-Order	OoO
HW Threads	1	1	1	2
Architecture	x86	x86	ARM-Like*	x86
Memory channels	4	4	-	2
Memory Freq.	1600/2400 DDR3/DDR4	1600/2400 DDR3/DDR4	2500 3D-stacked	2400 DDR4
Cache block	64B	64B	32/64B	64B
Technology	22 nm	22 nm	28 nm	14 nm
L1 Cache (L1D/L1I)	32K/32K 3 cycles Latency	32K/32K 3 cycles Latency	8K/8K 3 cycles latency	32K/32K 8-way set associative
L2 Cache	256K 10 cycles Latency 8-way set assoc.	256K 10 cycles Latency 8-way set assoc.	-	256K 4-way set associative.
L3 Cache	16M Shared 30 cycles Latency 16-way set assoc. 6 banks H3 Hash	16M Shared 30 cycles Latency 16-way set assoc. 6 banks H3 Hash	- -	2M 16-way set associative

*We simulate in-order cores with a similar performance to ARM cores.

The details of the three different computer architectures are shown in Table 4.1. DDR Setup 1 and DDR Setup 2 are similar to modern systems and include several Out-of-Order (OoO) cores (64 and 36 OoO cores respectively). These architectures will be evaluated using both DDR3 and DDR4 memory technologies onward in this chapter. The third architecture that we propose is an energy efficient and memory focused architecture with a 3D-stacked memory module. This setup can be considered as PIM, as the computing cores (small power-efficient cores) are allocated right under the 3D-stacked memory layers.

For the power-efficient small cores under 3D-stacked memory layers, we simulate the behaviour of ARM-like cores. The framework McPAT supports ARM architectures. However, ZSim does not support either ARM ISA and architectures. Therefore, with ZSim we have used x86 in-order cores (Zsim Simple in-order model [97]) configured at low frequencies (defined in Figure 4.1), with a performance comparable to commercial ARM-A35 [2] processors. On the memory side, the 3D-stacked cube is simulated using the HMC memory model included in Ramulator [103].

Typical modern architectures include several megabytes of cache memory spread between two or three levels. As previously mentioned, applications heavily

UNIVERSIDAD DE MÁLAGA

dependant on random memory access usually do not take advantage from typical cache configurations. Therefore, for area and power consumption optimization, we have heavily reduced the cache memories for the PIM setup. Specifically, we have removed L2 and L3 caches and reduced L1 caches to 8K (for both data and instruction caches), as 8K is enough for most random access applications while allowing us to keep a very low area and power consumption (detailed in next sections). Regarding the cache block size, we have considered both 64B and 32B for PIM setup, rather than just the typical 64B block size. Smaller cache blocks reduce the memory traffic between main memory and processor at the cost of worsening throughput for applications with sequential memory accesses.

For comparison purposes, we have also considered a real commercial system (Intel i7-8700 [52]). We have run experiments on both real i7-8700 and simulated i7-8700 architecture in order to establish a baseline and compare the results obtained from ZSim with those from the real machine. The details of this system are shown in Table 4.1.

4.2 Area and Power Consumption Estimation

When developing new computer architectures, area and power consumption are some of the most important factors to consider. Specially when we are working with systems like PIM, with important constraints in terms of both area and energy consumption.

In order to achieve an accurate power consumption and area of the cores of the different architectures that we analyze, we have used two different methods:

1. Using McPAT, an integrated power, area and timing modeling framework.
2. Using the reported area and power consumption specifications of real processors (which are similar to the simulated ones in the architectures that we analyze) to estimate the area and power of the aforementioned architectures.

Figure 4.1 shows all the estimated (in purple), simulated (in red) and reported (in green) power consumption for both real systems and our simulated setups. The grey top section of the bars shows the power consumption due to the memory system.

For the real commercial Intel i7-8700 [52], we are simply considering the specifications reported by Intel which are around 65 W of power consumption.

UNIVERSIDAD DE MÁLAGA

The details of the proposed architectures are elaborated in the next sections as separate cases depending on the considered setup.

4.2.1 PIM Setup Case

According to McPAT, each small PIM core at 1500MHz has around 2 mm^2 in area and has a power consumption of approximately 0.5 W per core at 28 nm technology, and 0.61 mm^2 of area and 0.204 W consumption per core at 22 nm technology (in red in Figure 4.1).

On the other hand, the cores included in our proposed PIM setup are very similar to the smallest A35 cores shown by ARM, but at higher frequencies. Considering this, we can estimate the area and consumed power based on the ones of the real ARM processors. Power consumption reported by ARM per core at 1 GHz frequency [35] is 90 mW (in green in Figure 4.1). Usually, power consumption is not linear with frequency. We consider reasonable an increase of 2 times the energy for each 1.5 times the frequency. Therefore, we can assume around 0.4 mm^2 in area per each small core and a conservative estimation for power consumption of 180 mW at 1.5GHz (in purple in Figure 4.1).

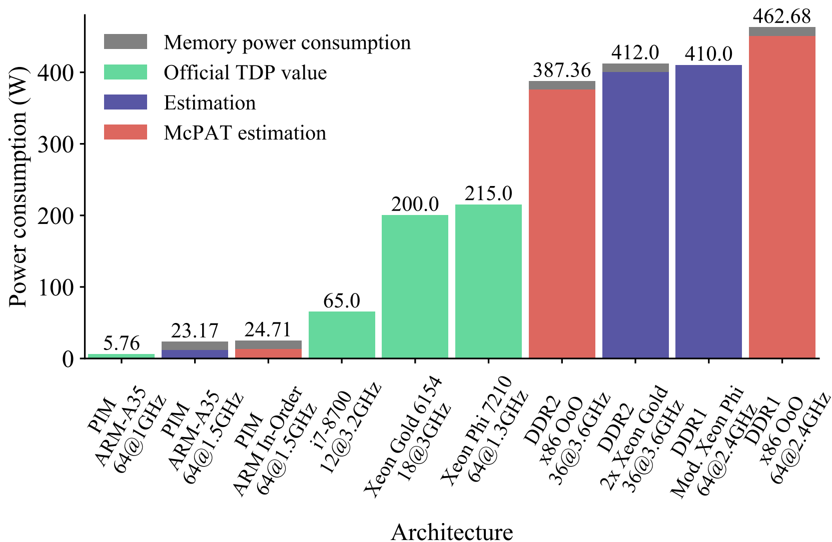


Figure 4.1: Processor power consumption for each architecture

UNIVERSIDAD DE MÁLAGA

All these figures of merit are the three first bars depicted in Figure 4.1 (note that each architecture has 64 cores). As mentioned before, the grey top section of the bars shows the power consumption due to the memory system. In this case, 3D-stacked memory power consumption has been estimated using Micron HMC power consumption calculator tool [82], giving a value of 11.65W.

4.2.2 DDR Setup Case

DDR setups with high performance cores do not have heavy area limitations. Therefore, we are not performing an in-depth analysis of area for them, but we do it for power consumption.

Estimations from McPAT for high performance cores give us a TDP (Thermal Design Power) of 450.68 W for the DDR Setup 1 (64 OoO cores at 2.4GHz) and 375.36 W for DDR Setup 2 (36 cores at 3.6GHz) shown in red in Figure 4.1.

As mentioned before, we also consider real systems power consumption. Some commercial processors with an architecture similar to DDR setups are:

- Intel Xeon Phi 7210 [115] with 64 cores at 1.3 GHz-1.5 GHz similar to DDR Setup 1.
- Intel Xeon Gold 6154 [114], with 36 cores at 3 GHz-3.7 GHz akin to the DDR Setup 2.

Between DDR Setup 1 and Xeon Phi there is a significant frequency difference (from around 1.4 GHz to 2.4 GHz), so we estimate that this setup should consume around 370 W and 450 W. In Figure 4.1 we have represented in purple the mean power consumption. We have to mention that in the case of Xeon Phi processor the memory system power consumption is included in the power estimation of the chip (hence the top section is not included in the bar).

In the case of DDR Setup 2 and a dual Xeon Gold processor, they should be quite similar, so our DDR Setup 2 should have a TDP of around 400 W (in purple in Figure 4.1).

Regarding to the power consumption of the memory system, DDR power consumption has been roughly estimated around 3 W per memory module, considered a reasonable value for DDR4 DIMM modules. These values have been added to the power estimations and represented (in grey) in Figure 4.1.

As mentioned earlier, we have included in the figure the reported power consumption of real processors (in green), in this case, an Intel Xeon Phi 7210 and

UNIVERSIDAD DE MÁLAGA

a Xeon Gold 6154.

4.3 Experimental Evaluation

We have evaluated the throughput of several benchmark and applications on the proposed architectures. The benchmarks and applications are:

- STREAM benchmark: a well-known benchmark used to measure sustainable memory bandwidth [80].
- RANDOM benchmark: this is a benchmark that we have developed in order to accurately measure the memory bandwidth when issuing random memory access. More details can be found in Appendix A.
- Several configurations of FM-index exact matching applications.

4.3.1 Setup and Methodology

The evaluation was conducted on the three different simulated architectures described in section 4.1 and a real machine (that was simulated as well). For the simulation, we have used ZSim [97] git public version [120] together with Ramulator-Pim public version [61, 103].

The public ZSim version has been modified to support direct communication with Ramulator. This has been done imitating the way that ZSim communicates with the DRAMSim2 memory simulator.

All experiments using PIM and DDR4 memory modules have been executed using the combination of Ramulator memory model and ZSim core model, as ZSim default memory model does not support DDR4 and HMC modules. Experiments including DDR3 modules have been run using both Ramulator and ZSim memory models for comparison purposes.

The platform where the experiments have been executed is a system including two Intel Xeon Gold 6154 with 18 cores at 3 GHz each and 384 GB DDR4 memory running Ubuntu 18.04.1. We used the GNU Compiler Collection (GCC), with common flags and `-O3` optimization, to compile the benchmarks and FM-index applications.

In addition, the experiments have been conducted on a real system Intel i7-8700[52] Coffee Lake processor with 6 cores (12 threads [110]) at 3.2 GHz each,

UNIVERSIDAD DE MÁLAGA

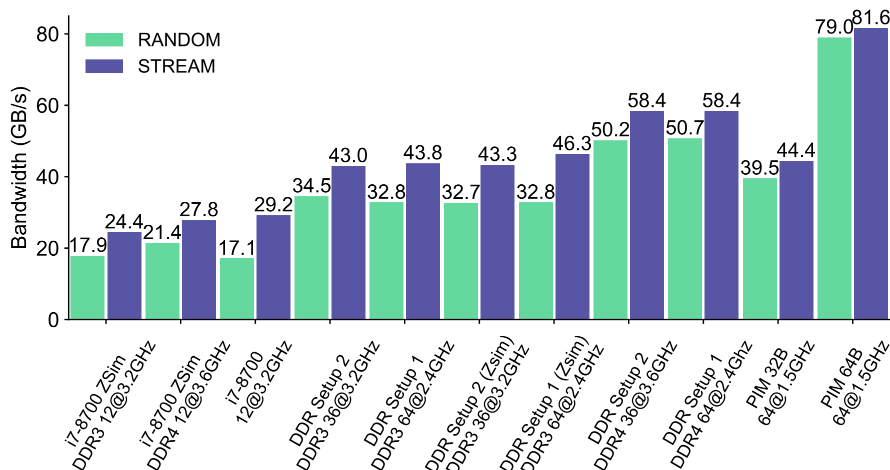


Figure 4.2: STREAM [80] and RANDOM memory bandwidth results for several architectures

able to reach 4.6 GHz using turbo-boost. It includes 64 GB of DDR4 memory and runs Ubuntu 16.04.6. Applications and benchmarks have been compiled using the GNU Compiler Collection with common flags and `-O3` optimization.

Specific parameters of the experiments are:

- STREAM benchmark process an array size of 10,000,000. Reported values are the maximum of all STREAM results.

- RANDOM benchmark generates 1024 MB data structure, with 3,000,000 random accesses.

- FM-index applications searches 500,000 sequences with around 200 bases each in a reduced 1 GB genome.

All benchmarks and applications have been run on all the available threads in each system. Experiments have been run at least three times and we have reported the average value of them.

UNIVERSIDAD DE MÁLAGA

4.3.2 Results of the Benchmarks

Figure 4.2 shows the maximum memory bandwidth obtained from both STREAM and RANDOM benchmarks. Clearly, the performance for PIM setups is much better, achieving between 2.7× and 3.4× higher throughput when compared with i7-8700 architectures and around 1.4×-1.9× higher throughput for the 36 and 64 multicore systems.

We can also observe that RANDOM results are much closer to STREAM results for PIM architectures, as these ones do not suffer of the higher latency and deeper cache architectures of the typical DDR architectures.

Figures 4.3, 4.4 and 4.5 show different metric results from the RANDOM benchmark for different arithmetic intensities. For each figure, top graph shows the results when accessing a single block per step and bottom part shows the results for accessing two consecutive blocks per step.

PIM architectures get better results for lower arithmetic intensities, but they are worse than DDR architectures for high arithmetic intensities because of the lower frequency and lower computing power of the in-order cores. This can be specially appreciated in Figure 4.5, which shows the amount of integer operations performed per second.

If we look at the results when we load two consecutive blocks, we can observe that bandwidth results (Figures 4.3 and 4.4) improve for higher amount of operations, but the operations per second (Figure 4.5) decrease for the setups more limited in memory. This can be easily explained because we are loading twice the data for the same amount of operations.

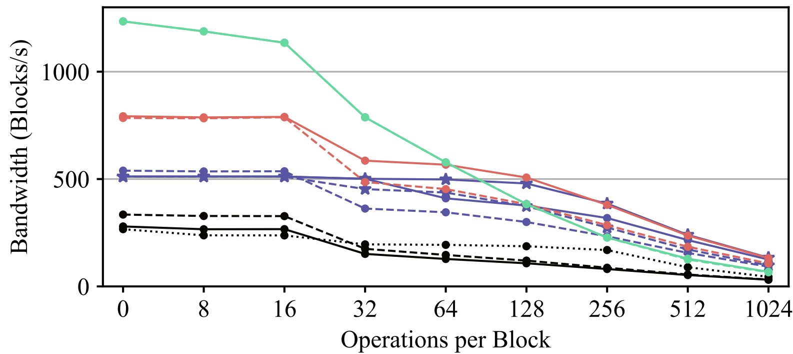
Figure 4.4 shows that the PIM architecture with 32 bytes cache line reach around half the memory bandwidth of the 64 bytes one. This sequential bandwidth limitation does not affect any other benchmark because other applications do not use more than 32 bytes from each block brought from main memory to the processor.

For simulator validation, we can compare real and simulated i7-8700 setups. Results are reasonably similar, with some small differences:

- Figures 4.3, 4.4 and 4.5 show that real i7-8700 achieves slightly worse bandwidth for low arithmetic intensities and slightly better for higher ones.
- On the other hand, Figure 4.4 shows that the real system achieves slightly better throughput for Stream benchmark, but slightly worse for Random benchmark.

UNIVERSIDAD DE MÁLAGA

Those small differences could be explained because of the differences between real and simulated architectures imposed by simulation setup restrictions, like the memory system frequency limitations and the absence of Intel Turbo-boost technology.



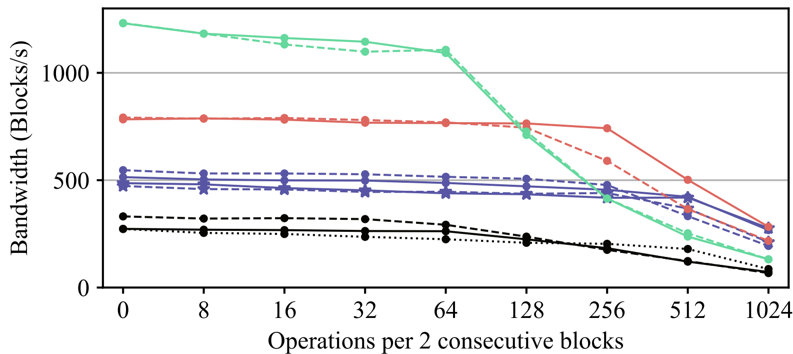


Figure 4.3: Benchmark memory bandwidth in memory blocks per second for different arithmetic intensities. (*) Marked setups use Zsim memory model

UNIVERSIDAD DE MÁLAGA

4.3.3 FM-index Results

We have chosen the FM-index based exact matching as a real application to be evaluated with random memory access patterns. We have compared three different versions of the algorithm: Basic FM-index [33] (k1d32), 2-Step FM-index [23] (k2d16) and Bit-vector 2-step FM-index [57] (k2d64bv). The details of these different versions can be found in chapter 3.

Figure 4.6 shows the throughput (in terms of Giga-LF operations per second) for the different architectures and FM-index versions. PIM architectures achieve

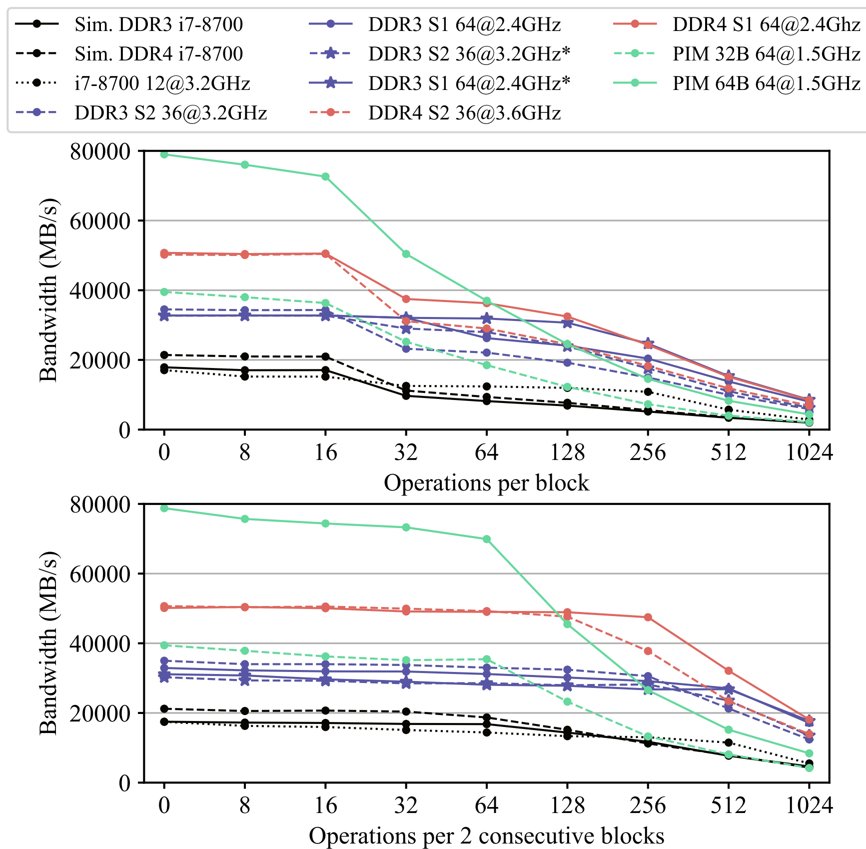


Figure 4.4: Benchmark memory bandwidth for different arithmetic intensities. (*) Marked setups use Zsim memory model

UNIVERSIDAD DE MÁLAGA

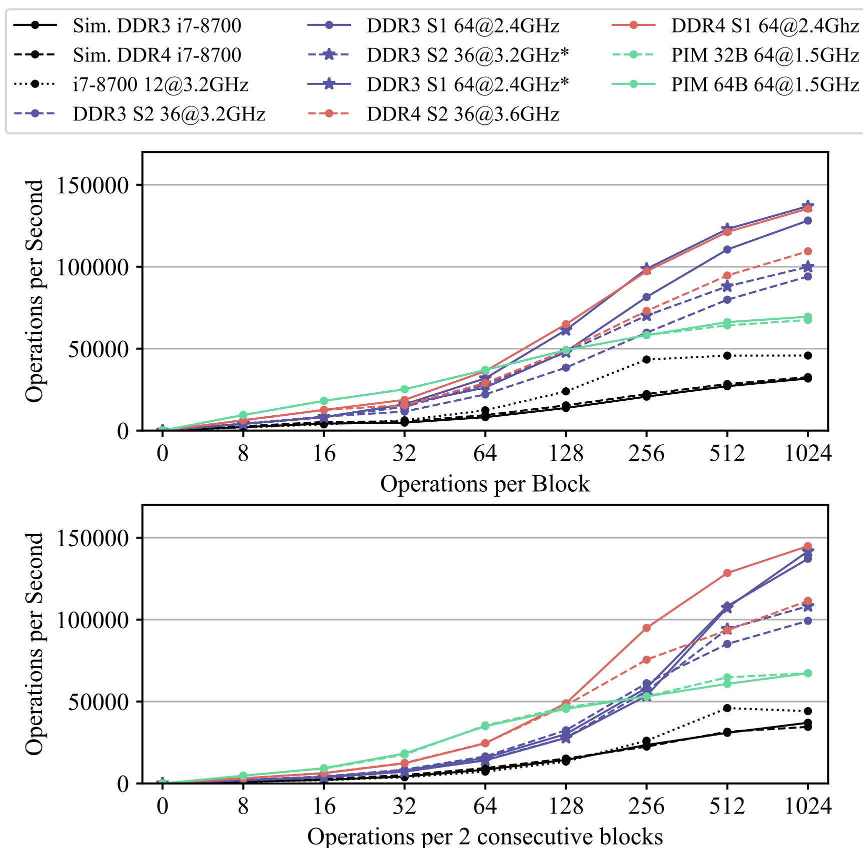


Figure 4.5: Benchmark operations per second for different arithmetic intensities. (*) Marked setups use Zsim memory model

around 2.7×-3.7× higher throughput than 12 cores setups and between 1.26× and 1.87× more throughput than 36 and 64 cores setups with both DDR3 and DDR4 memories. From the figure we can observe that DDR4 gets also around 25% more throughput than DDR3 memories. As expected, the split bit-vector 2-step version (k2d64bv) is able to perform around twice the amount of LFM operations per second than other FM-index versions. Finally, let us note that PIM architecture using 32 bytes cache lines gets almost the same performance as the one using 64 bytes cache lines. This is easily explained because those versions of FM-index does not use more than 32 bytes from each block.

UNIVERSIDAD DE MÁLAGA

4.3.4 Roofline Model

As stated in section 2.8, the Roofline model defines arithmetic intensity as the number of floating point operations per loaded byte from main memory. However, since our application does not perform floating point operations, we consider any kind of instruction instead of just floating point operations for our arithmetic intensity.

Figures 4.7, 4.8, 4.9, and 4.10 show the roofline model for the intel i7-8700 system and for each simulated setup.

Horizontal line shows the computing limit for each architecture (both real and simulated in the case of i7-7800 model). It has been measured with the RANDOM benchmark, and a big amount of integer operations per each loaded block from memory (4096 operations per block) in order to not be bounded by the memory bandwidth. Diagonal lines show the bandwidth limit, both for sequential accesses (measured with STREAM) and for random accesses (purple solid and dashed lines respectively). As shown in previous sections, random accesses bandwidth are always lower than sequential ones, being the differences almost non-existent on PIM architectures.

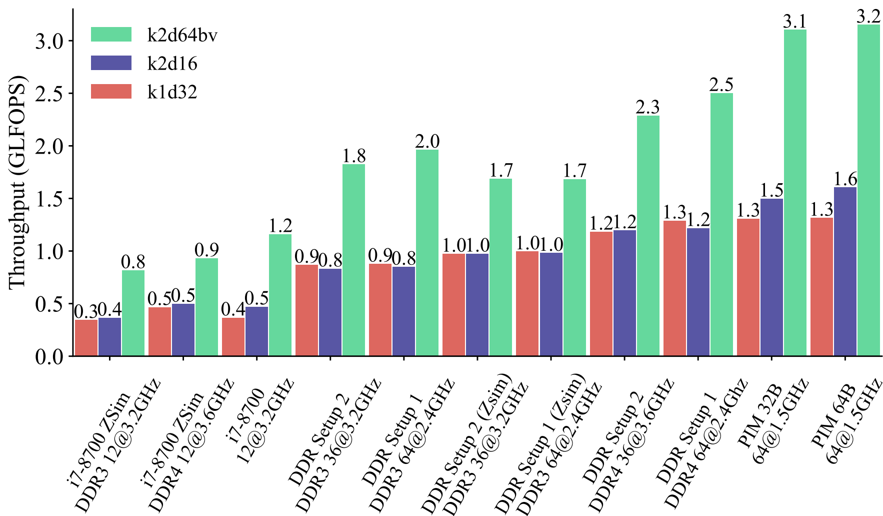


Figure 4.6: FM-index throughput for several architectures

UNIVERSIDAD DE MÁLAGA

Roofline figures 4.7, 4.8, 4.9, and 4.10 show how our RANDOM benchmark fits well to the roofline model, specially for the real i7-7800 setup. DDR Setups get a better performance for high arithmetic intensities (up to almost 2× instructions per second). On the other hand, the PIM setup has a much better performance for low arithmetic intensity, and a higher memory bandwidth. This can be appreciated on the bandwidth limits lines, much more close to the vertical than the DDR ones.

i7-7800 shows some difference between real and simulated system, this can be because of the higher turbo-boost processor frequency rate supported for the real system, which can not be simulated on ZSim.

4.3.5 Power efficiency

Providing McPAT with the power consumption data presented in Section 4.2 and the zsim output stats, we have obtained the efficiency for each setup measured in operations performed per consumed joule. Figure 4.11 shows the efficiency for both RANDOM benchmark and k2d64bv FM-index version.

We can observe a great improvement for the PIM architectures when com-

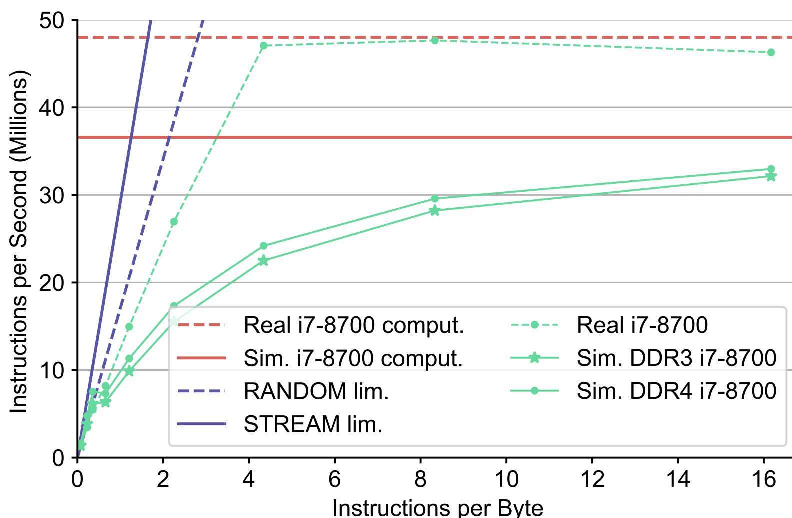


Figure 4.7: i7-8700 Roofline model

UNIVERSIDAD
DE MÁLAGA

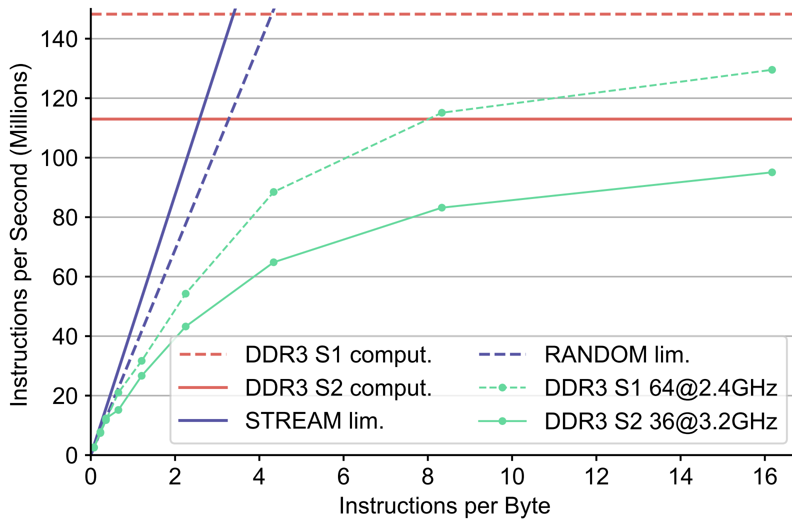


Figure 4.8: DDR3 Roofline model

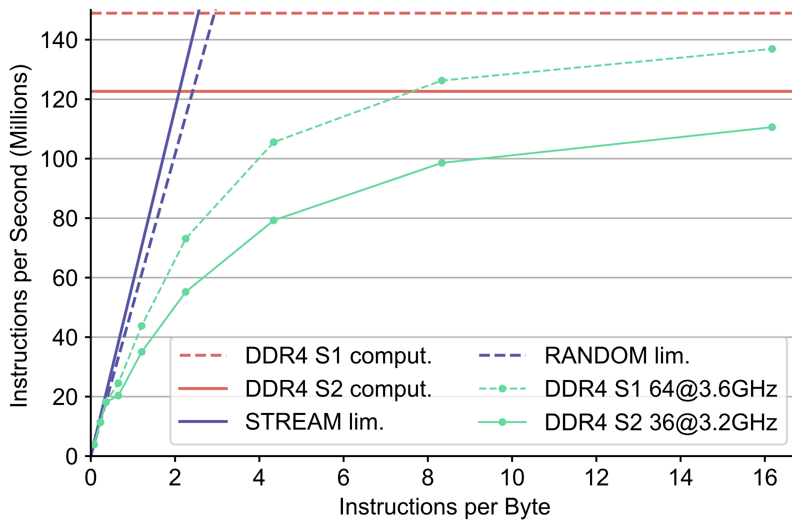


Figure 4.9: DDR4 Roofline model

UNIVERSIDAD DE MÁLAGA

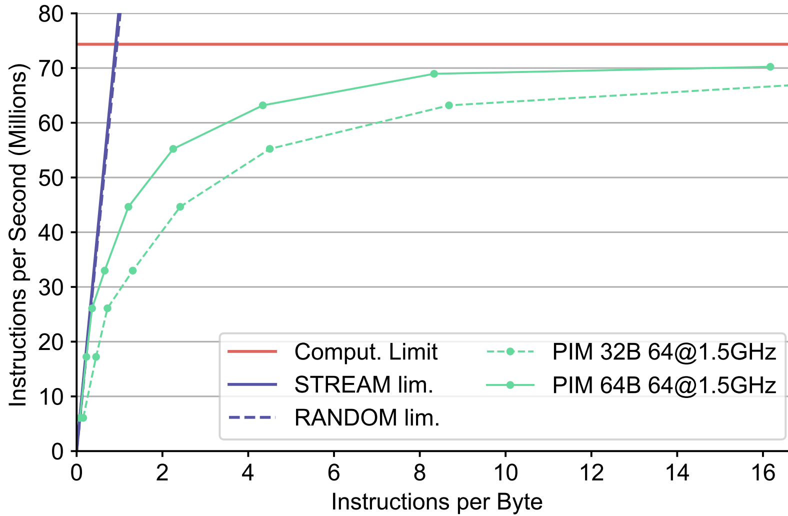


Figure 4.10: PIM roofline models

pared with conventional DDR memory systems, being able to perform around 8× more LF operations and random accesses than i7-8700 real system. We can observe, as well, that PIM architectures get even greater improvement than bigger systems with more complex cores, using around 21× less energy per operation.

4.4 Related work

During last years, new memory technologies like Near-Data-Processing and specially Processing-In-Memory are gaining importance [41], in order to solve the problems derived from the memory wall [113] and data-intensive applications.

Consequently, a significant amount of works around these topics has appeared during the last years. Most of them, like ours, based on the Micron HMC[81] architecture, expanding or completely reworking the logical layer. For example, some works based on HMC are [6] and[5], oriented to optimize Google PageRank and parallel graph processing respectively; [19], analyzing the performance of google workloads; [84] and [119], optimizing graph processing applications; [47], an analysis of MapReduce workloads on HMC; [29], presenting a Near-Memory-Processing accelerator for basic data analytic operators. The work presented in

UNIVERSIDAD DE MÁLAGA

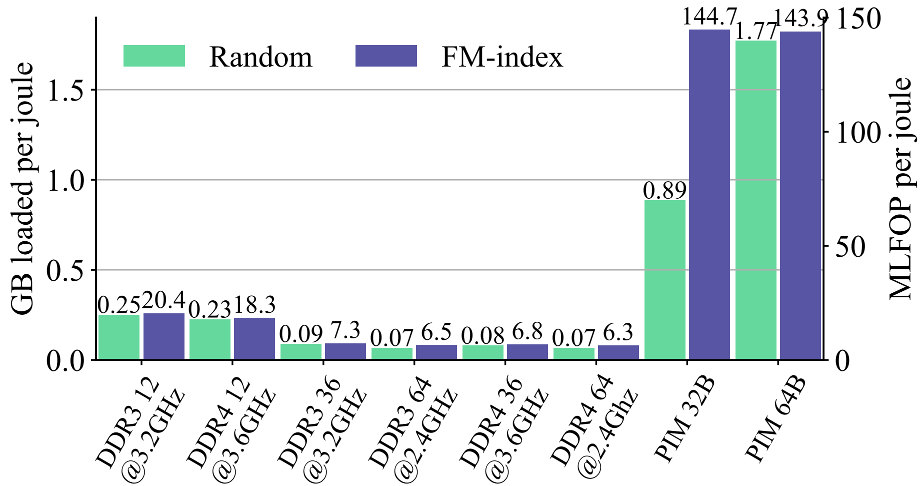


Figure 4.11: RANDOM memory bandwidth and LFOPs per Joule for different architectures

[39], includes general near-data processors in the logic layer and analyzes their performance for common applications like MapReduce, PageRank and Neural Networks. Others, such as [40] and [59], are focused on Neural Networks acceleration; and [60], with some common points with our work, improves bioinformatics applications performance through PIM architectures; in [55], it is analyzed the density and performance of HMC and [20], proposes a cache coherence protocol for near-data accelerators.

Furthermore, some works are oriented to use different architectures, like [50], working with Near-Data-Processing on GPUs; or [117], mixing CPUs and GPUs close to the data; and [30], introducing Computational RAM; as well as [31] and [11], implementing these techniques with commodity DRAM modules and [48], introducing a new architecure called DIVA.

4.5 Conclusions

This chapter presents an analysis and research of several architecture with novel memory architectures, specifically *Processing-In-Memory* and *Near-Data-Processing*,

UNIVERSIDAD DE MÁLAGA

which improve the throughput of workloads using random memory access significantly.

We have developed new architectures, optimized for these type of workload and compared them, using both real and simulated typical systems and architectures.

Processing-In-Memory setups, using a 3D-stacked memory with power efficient cores have been shown to achieve a much better performance and a much lower energy consumption than conventional architectures with DDR-type memory technologies.

As explained in depth in section 4.3, our PIM setups achieve more than 3× the throughput of RANDOM benchmark and FM-index algorithms and in some cases, with an energy efficiency up to 40×, compared with systems using typical DDR memory technologies and deep cache hierarchies.

UNIVERSIDAD DE MÁLAGA

5 Bowtie2 on Processing in Memory Architectures

Our lives are increasingly reliant on new technologies, most of them based on big data centres and huge high performance computing (HPC) nodes. This kind of computing systems is used for a lot of different disciplines with a great impact on our lives, from scientific ones (like particle physics, material sciences, genomics, precision medicine, etc) to applications we use every day (like weather forecast, global communications, entertainment services). However, this has also a great impact on the global power consumption. In 2018, the global data center energy demand was 198TWh, around 1% of the global energy demand [1].

With the recent and future development of exascale systems, power consumption appears as one of the greatest challenges [12] for HPC systems, being constrained at 20MW [101]. Typically, HPC systems use float operations per second (FLOPS) as performance measurement unit, however, this is not very representative for some workloads, more affected by the memory-processor communication limits than by the raw computation power. Some examples of memory-bound applications are pointer chasing [51], graph processing [102], and some bioinformatics applications.

Next-Generation-Sequencing (or NGS) is a name used to refer to the new, high performance methods and technologies used to sequencing both ADN and ARN base pairs. It has a great impact in several fields, like genomics, cancer, medical, food microbiology, precision medicine and drug discovery research [98, 17, 94, 118]. DNA sequence alignment is an example of relevant memory bounded application. This application is usually based on full-text index, like FM-index, which presents a random and unpredictable memory access pattern, similar to some pointer chasing applications. These applications are great candidate for

UNIVERSIDAD DE MÁLAGA

analysis and optimization focused in energy saving with low-power processors and high bandwidth memory technologies.

In this chapter, we present architectural exploration for random memory access applications, specifically using Bowtie2, a popular sequence alignment application, as the case study, for energy efficient computation. Since these applications are memory bounded, we propose to use energy efficient ARMv8 64-bit cores. In addition to the compute side we also propose to use 3D-stacked high bandwidth memory (HBM2) [106] instead of traditional DDR memories like DDR4. We compare the performance of ARM based system to that of Intel Xeon Phi 7210 KNL processor [104], which has an integrated stacked 3D MCDRAM on the package. The exploration is performed using gem5-X [90] which is an extended and validated version of widely used gem5 architectural simulator [15] with validation error of up to 4%.

The main contributions of this chapter are as follows:

- We get 69% performance and 71% energy benefit when using HBM2 instead of DDR4.
- We demonstrate that using many ARMv8 in-order cores, results in 2x more energy savings when compared to ARM out-of-order (OoO) cores.
- We demonstrate that scaling the number of cores and frequency results in 8x energy efficiency.
- We demonstrate that 16 ARM OoO cores outperform 32 threads running on the Intel Xeon Phi KNL.

5.1 Related work

Most HPC infrastructures and data centers are based on typical x86 CPUs along with GPUs, with some scarce exceptions like IBM POWER or ARM architectures. Since it was released, Intel's Xeon Phi KNL has been increasingly adopted by the HPC community [104, 89], mainly due to its high parallelism, high scalar and vector performance and the 3D-stacked MCDRAM memory, able to achieve more than 400GB/s of peak memory bandwidth.

However, recent studies like [91, 92, 100, 67] have evaluated the use of energy-efficient ARM cores in HPC domain. Some works, like [92, 100, 67, 87], state that ARM based systems are more energy efficient than x86 based systems, but they achieve a lower raw performance and computing power, making them not suitable

UNIVERSIDAD DE MÁLAGA

for HPC workloads. Also, the authors in [91] oppose this opinion, suggesting that ARM based systems are neither energy efficient nor high performing as compared to current x86 systems. All those researches have been performed using compute-bounded benchmarks, with x86 clearly outperforming ARM based systems.

Other papers, like [91, 92], analyze the memory bandwidth of ARM based systems with traditional DDR modules as main memory when running memory bounded applications and benchmarks. Those works conclude the ARM systems fall behind x86. On the other hand, the authors in [92] suggest that ARM based systems could be benefited from future 3D high bandwidth memories.

Bowtie2 is a memory bounded application with random memory access pattern, based on the FM-index algorithm and data structures. Related to this, previous studies [66] have been focused on parallelism using many-core Intel's Xeon Phi KNL. In addition to KNL, other typical HPC systems, like GPUs are also used for bioinformatics workloads, as in [112]. Also, in [45], authors utilize HPC type many-core x86 clusters for NGS. All these studies indicate that genome sequencing is usually performed by high performing power hungry HPC resources.

To the best of our knowledge, no studies have been conducted using ARM based architectures along with 3D-stacked memories for memory bounded HPC workloads, such as NGS. In this chapter, we demonstrate that random access memory bounded workloads can be executed on energy efficient ARM based platforms, with performance at par or surpassing that of existing x86 or accelerators like KNL, provided there is high memory bandwidth available such as the 3D-stacked HBM2.

5.2 Sequence Alignment Application: Bowtie2

Bowtie2 [64] is an open-source, ultra-fast and memory-efficient alignment application used for aligning DNA reads to large genomes, able to support gapped alignments. It relies upon BWT and FM-index algorithm (see section 2.2 and Figure 5.1.(a)) to quickly find non-exact alignments that satisfy a specified alignment policy. Bowtie2 algorithm is divided into four steps (Figure 5.1.(b)):

- Bowtie2 extracts "seeds" substrings from each read.
- Those seeds substrings are aligned to the genome, without supporting gaps, reporting the BWT ranges of the occurrences.

UNIVERSIDAD DE MÁLAGA

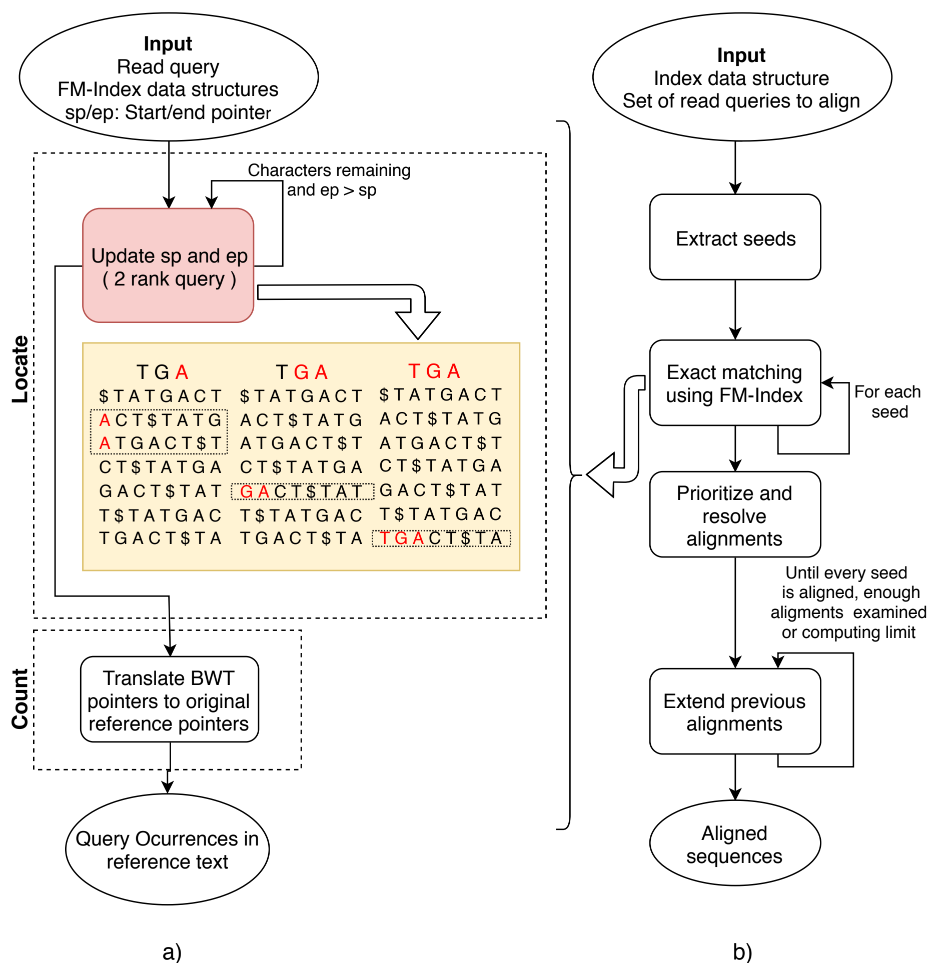


Figure 5.1: Application phases for (a) FM-index and (b) Bowtie2

- Seed alignments are prioritized, calculating their positions in the reference genome using the FM-index.
- Prioritized alignments from step 3 are extended into full alignments, considering gaps. This last step is performed using SIMD-accelerated dynamic programming.

Bowtie2 indexes are optimized in order to use as less memory as possible. This way, a Bowtie2 index for the human genome uses around 3.24GB on disk,

UNIVERSIDAD DE MÁLAGA

and has a memory footprint of just 1.3GB.

Compared to other sequence alignment tools, Bowtie2 is around 2.5-3x faster than Burrows Wheeler Aligner (BWA) when both applications are searching for gapped alignments. Therefore, we have chosen Bowtie2 to analyze the performance of the PIM architectures based on ARMv8 energy efficient processors.

5.3 Simulation Framework and Parameters for Architectural Exploration

Simulation framework enables us to perform fast architectural exploration for performance-energy optimized architectures for any given application. For HPC applications, we need a simulation framework capable of running multi-threaded applications on many-core simulated systems.

5.3.1 Experimental Setup

We use gem5-X [90], a validated and extended version of cycle-accurate gem5 architectural simulator [15], which exhibits a validation error of up to 4% when simulating ARMv8 64-bit cores. ARM full-system (FS) simulation mode is used with Ubuntu 16.04 as the OS and Linux kernel v4.14. FS mode is used to have a complete picture of the system, with full software stack, and also because Bowtie2 requires the multi-threading support only available in FS. Both ARMv8 64-bit in-order and OoO cores are used for the architectural exploration with L1 instruction (L1-I) and L1 data (L1-D) cache fixed at 32KB using ARM JUNO platform [9] as the starting point. Main memory of 4GB is used with both DDR4 and HBM2 in gem5-X.

For energy evaluation, we use the power model for 28nm CMOS bulk technology node for ARM Cortex A57 OoO core and ARM Cortex A53 in-order cores, as proposed in [88] and [90]. The power model includes the core active, wait-for-memory (WFM) and static energy. It also includes the LLC read/write and static energy as well. For the memory power models, we use the DRAM power values as reported in [68].

We have also performed experiments on a real system with an Intel Xeon Phi 7210 processor (KNL), 16 GiB of MCDRAM and 192 GiB of DDR4 running Ubuntu 16.04.1 Linux.

UNIVERSIDAD DE MÁLAGA

5.3.2 Methodology

In the this section, we will discuss different architectural parameters we change and explore to get an optimized architecture, as they have the most impact on performance (in terms of execution time) and energy of the system. The parameters we explore are:

High Bandwidth Memories: Memories with high bandwidth like high bandwidth memory (HBM2) [106] help in alleviating the memory bottleneck in memory bounded application. We propose to use HBM2 for such workloads. It is a 3D-stacked memory with a bandwidth of 307.2 GB/s [106], as it is available in gem5-X [90] with 8 independent channels. Energy values as in [86] are used for HBM2.

Core Type: We investigate how different core types like the ARMv8 64-bit in-order cores and OoO cores affect energy efficiency and performance.

Core Count: We explore how performance and energy scale with the number of cores. As the workload is parallelizable, we launch one thread per core. We also explore if using many in-order cores is beneficial both in terms of performance and energy, as compared to fewer OoO cores. We vary the number of cores from 8 to 28. We do not go beyond 28 cores as the simulations time drastically increases with the number of cores and the scaling trend can already be captured with 28 cores simulation, except for two cases with in-order cores, which we will discuss in section 5.4.

Core Frequency: We also vary the core frequency, and look into the scaling of in-order and OoO core frequency along with the number of cores, analysing the effects on energy and performance. We make frequency range between 1GHz to 2GHz as the extreme points.

Last Level Cache (LLC): We explore the effects on performance and energy of changing LLC size. We size the LLC along with number of cores as shown in Table 5.1. We first set the LLC size to 1MB irrespective of the core count. We then change the LLC size according the number of cores, so to have the same LLC size-to-core count ratio. We use two ratios, 1MB/8-cores and 2MB/8-cores. The ratio scales well with 8 and 16 cores, but for 24 and 28 cores, according to the ratio of 1MB/8-core ratio LLC size should be 3MB and 3.5MB, respectively. Since these sizes are not a power of 2, we scale up the LLC to 4MB for 24 and 28 cores. Similarly, for 2MB/8-cores, we use a size of 8MB LLC for both 24 and 28 cores. We also look into systems with no-LLC and how this affects both the performance and

UNIVERSIDAD DE MÁLAGA

energy consumption. Our experimental setups use one or two cache levels, considering L2 cache as LLC. This seems reasonable for ARM architectures and allows us to keep the system as simple as possible, reducing gem5-X simulation time, processor area and power consumption.

Memory Type: As memory is the main bottleneck resource in memory-bounded applications, we explore how HBM2 helps in alleviating this bottleneck and what benefits we get when compared to DDR4, for all the above architectural combinations.

Table 5.1: LLC Sizes and scaling with number of cores.

Core Count	**Fixed Size LLC**	**LLC 1MB/8-cores**	**LLC 2MB/8-cores**
8 cores	LLC = 1MB	LLC = 1MB	LLC = 2MB
16 cores	LLC = 1MB	LLC = 2MB	LLC = 4MB
24 cores	LLC = 1MB	LLC = 4MB	LLC = 8MB
28 cores	LLC = 1MB	LLC = 4MB	LLC = 8MB

5.4 Results and Discussion

All the experiments described below have been performed lauching Bowtie2 in gem5-X simulator. Each test has performed 0.2 million read alignments, considering the execution time of the FM-index search algorithm within Bowtie2. The energy consumption results correspond to the energy for the complete system, including CPU cores, caches and memory during the Bowtie2 execution.

5.4.1 HBM2 vs DDR4

We first look into the performance and energy benefit of using HBM2 instead of DDR4, comparing the same system except for the memory. Figure 5.2 shows that HBM2 always gets better performance than typical DDR4 systems. At the same frequency, OoO cores achieve higher performance benefit than in-order cores. Performance benefit also scales well and increases with the increase in the core count with fixed L2 size or fixed L2-core count ratio, because the increase in the core count stresses more the memory channels and memory bandwidth of the system, much higher for HBM2 than for DDR4.

Energy efficiency scales in a similar way than performance (Figure 5.3), being always better for HBM2 than DDR4 and with a similar trend for both core count

UNIVERSIDAD DE MÁLAGA

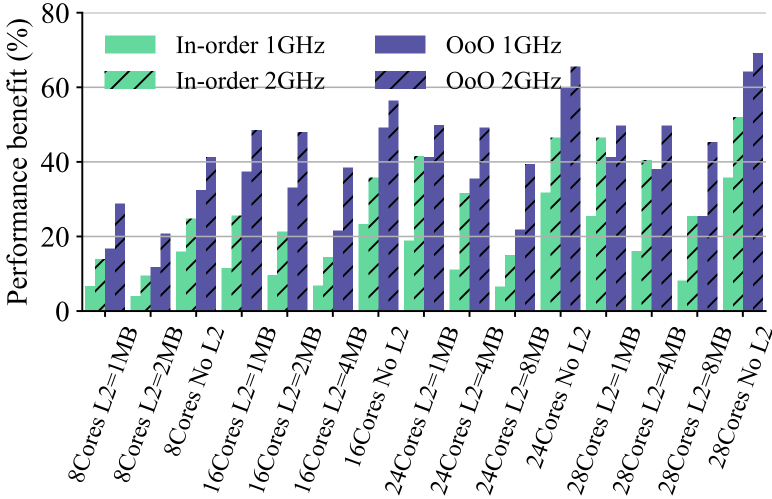


Figure 5.2: Performance benefit of HBM2 vs DDR4 with same cache hierarchy

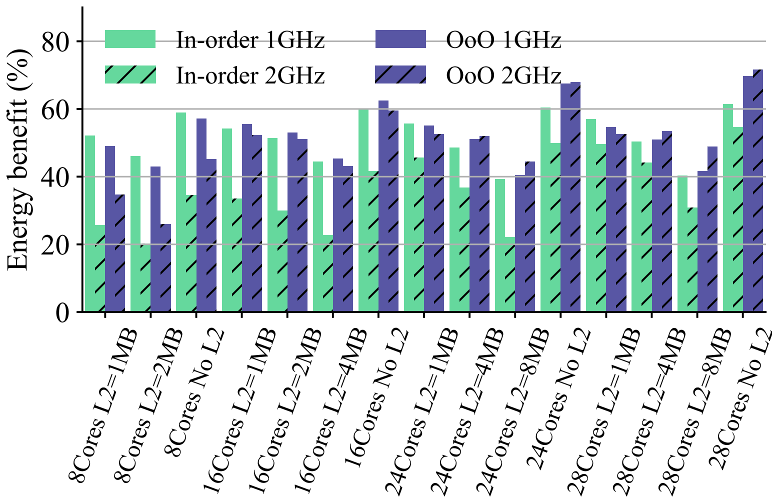


Figure 5.3: Energy benefit of HBM2 vs DDR4 with same cache hierarchy

UNIVERSIDAD DE MÁLAGA

and core type. However, in-order cores energy efficiency gets more percentage benefit of using HBM2 over DDR4 at 1GHz than when operated at 2GHz. Out of order cores present a similar trend for lower core count up to 16 cores, but it turns over with higher core count and more L2 cache.

From Figures 5.2 and 5.3 it can be seen that there is a performance benefit of up to 69% and energy benefit of up to 71.5% in a system without L2 and up to 50% performance and around 56% energy benefit in a system with L2.

5.4.2 Near Compute HBM2 (no L2) vs DDR4

As HBM2 is a 3D-stacked memory, it is packaged on the same die as the processor core, in contrast to DDR4, which is usually a separate package. Hence, in this section we look into the near-memory computation using HBM2 near to CPU core with no L2 and compare it to a DDR4 system with L2.

Figures 5.4 and 5.5 show percentage performance and energy benefit of HBM2 in a no-L2 system as compared to DDR4 with L2 in the system.

We can see that OoO cores with no-L2 HBM2 always are better than DDR4 with L2 both in terms of performance (by up to 68%) and energy (by up to 71.5%),

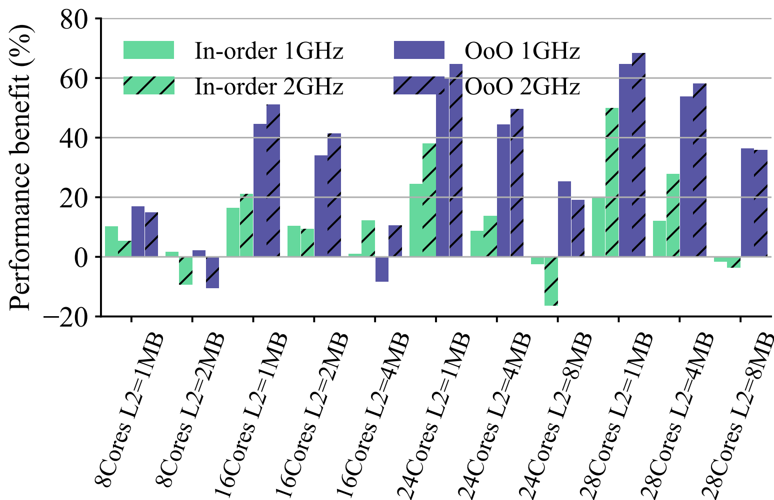


Figure 5.4: Performance benefit of HBM2 with no L2 vs DDR4 with L2

UNIVERSIDAD DE MÁLAGA

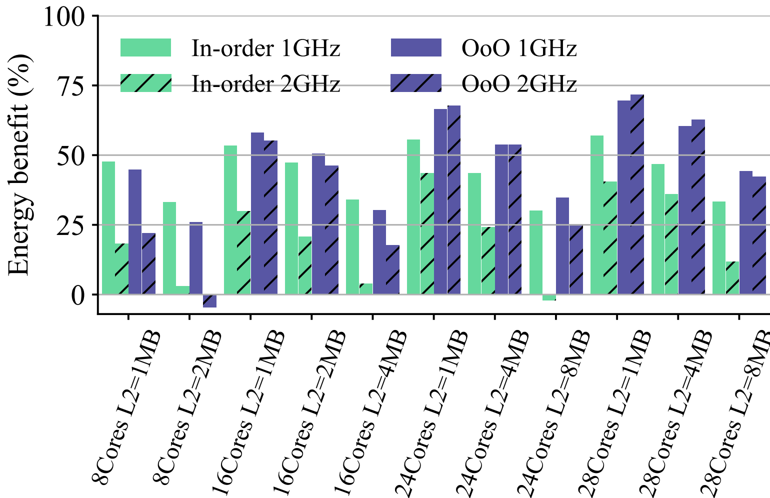


Figure 5.5: Energy benefit of HBM2 with no L2 vs DDR4 with L2

except when the number of cores are 8 and L2 is 2MB, as in this configuration, the cores have quite big L2 which helps in hiding away the latency to the memory. The same trend is true for energy benefit for OoO cores. For in-order cores, the performance and energy benefit increase with number of cores, except for the configuration, when the number of L2-to-core ratio is larger, leading to slightly negative, performance benefit. In that case, since in-order cores cannot stress the HBM2 memory bandwidth sufficiently, L2 helps in hiding away the latency to the memory. However, energy benefits still remain, even if the performance benefits are negative.

5.4.3 Performance-Energy Scaling with Core Count

In this section, we explore the scaling of performance and energy with the number of cores, both for in-order and OoO cores, with different cache sizes, as well as with both HBM2 and DDR4.

Figures 5.6 and 5.7 show the performance and energy scaling at 2GHz, respectively. The red horizontal line corresponds to the execution time for this KNL processor. Firstly, we see that performance improves when increasing the number of cores, except for OoO cores with LLC. This is because, higher number of cores

UNIVERSIDAD DE MÁLAGA

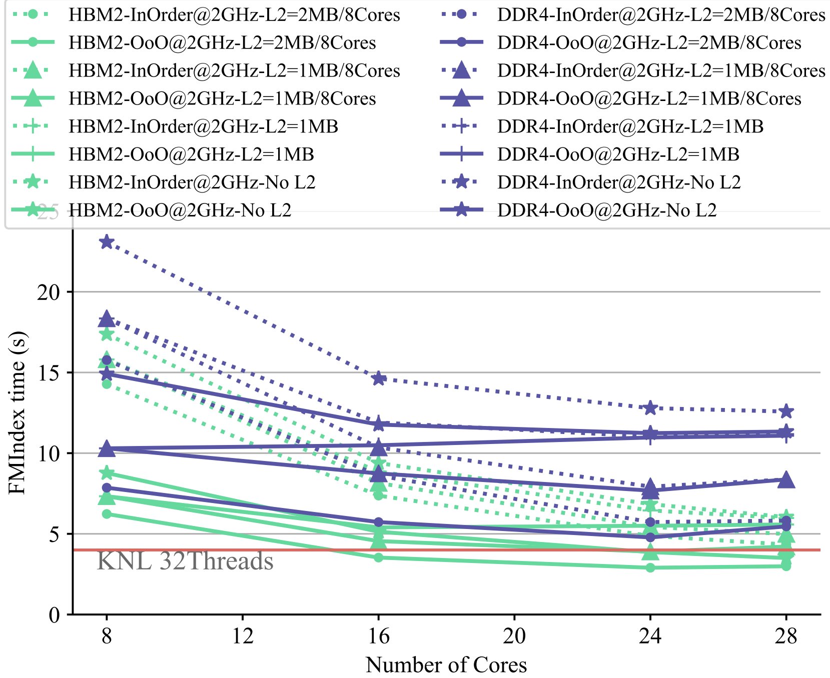


Figure 5.6: Performance scaling at 2GHz

implies more memory request through LLC, causing a bottleneck. Systems with in-order cores do not show this effect, as these cores are not able to generate as much memory request as OoO cores.

Secondly, we can see how several configurations in Figure 5.6 either match or outperform the performance of state-of-the-art 32 KNL cores. For example, 32 ARM in-order cores at 2GHz match the performance of 32 KNL cores. Also, we observe that many in-order cores can match or outperform fewer OoO cores performance with much lower energy consumption.

For example, performance of 8 OoO cores with HBM2 and no L2 is outperformed by 24 in-order cores with HBM2 and no L2, by 22%, with 37% less energy as it can be seen in Figure 5.7. If we consider area, single OoO core area for 28nm CMOS bulk ($2.05mm^2$) is almost 3 times that of single in-order core ($0.7mm^2$) as reported in [90, 36, 37]. Hence within the same area budget, in-order cores outperform OoO both in terms of performance and energy. If we look at Figures 5.6 and 5.7, we can find more cases where many in-order cores outperform or

UNIVERSIDAD DE MÁLAGA

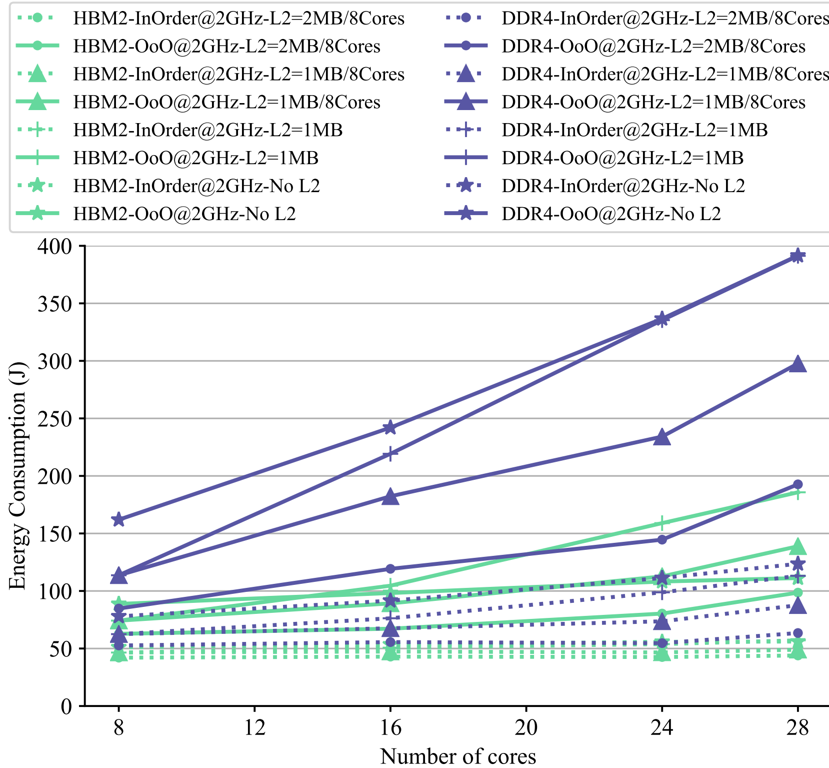


Figure 5.7: Energy scaling at 2GHz

match the performance of fewer OoO with up to 50% less energy.

5.4.4 Performance-Energy Scaling with Frequency

In this section, we also explore the performance and energy scaling at different core frequencies. Figures 5.8 and 5.9 show the performance and energy scaling at 1GHz, respectively (in addition to the results at 2GHz discussed in section 5.4.3). As previously mentioned, the red horizontal line corresponds to the performance for this KNL processor and it will be discussed in section 5.4.5. We observe that the performance and energy trend at 1GHz is similar to that at 2GHz. We can observe that similar to 2GHz, at 1GHz also many in-order cores can outperform fewer OoO cores, both in terms of performance and energy, with equal or less

UNIVERSIDAD DE MÁLAGA

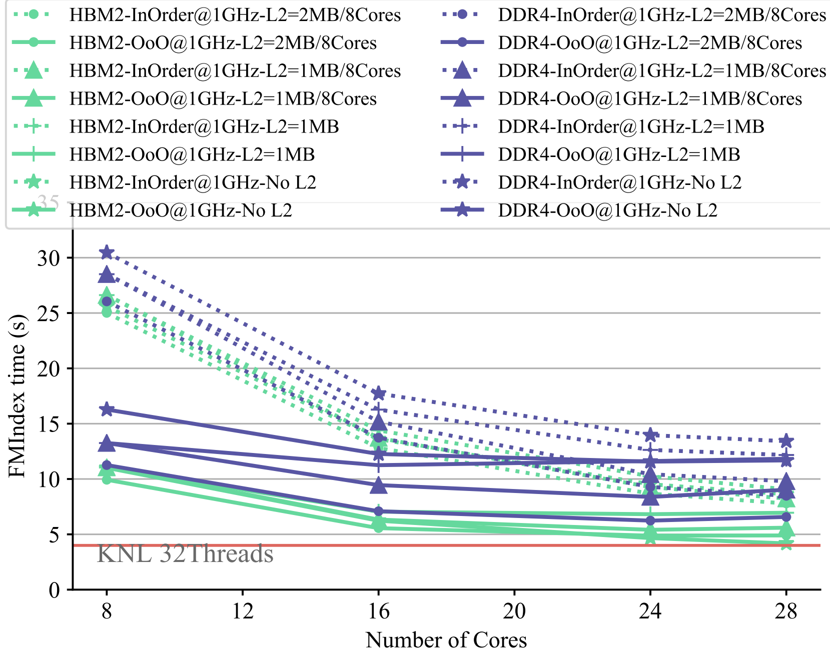


Figure 5.8: Performance scaling at 1GHz

area footprint. We also observe that many in-order cores at 1GHz match the performance of fewer OoO cores at 2GHz. For example, 28 in-order cores at 1GHz with L2=2MB/8 cores and HBM2 match the performance of 8 OoO cores at 2GHz with L2=1MB/8 cores and HBM2, giving an energy efficiency of 8x, with an area overhead of 20%.

5.4.5 Comparison to Intel Xeon Phi KNL

We have also compared the performance of the application Bowtie2 in an Intel Xeon Phi Knights Landing (KNL) proccessor (64 threads at 1.5Ghz, detailed in section 5.3.1) with the different in-order and OoO ARM cores configurations. Experiments on KNL processor have been performed using both conventional DDR4 memory and the high bandwidth memory available for Intel Xeon Phi KNL processors.

Figures 5.6, 5.8 and 5.10 show a red horizontal line corresponding to the

UNIVERSIDAD DE MÁLAGA

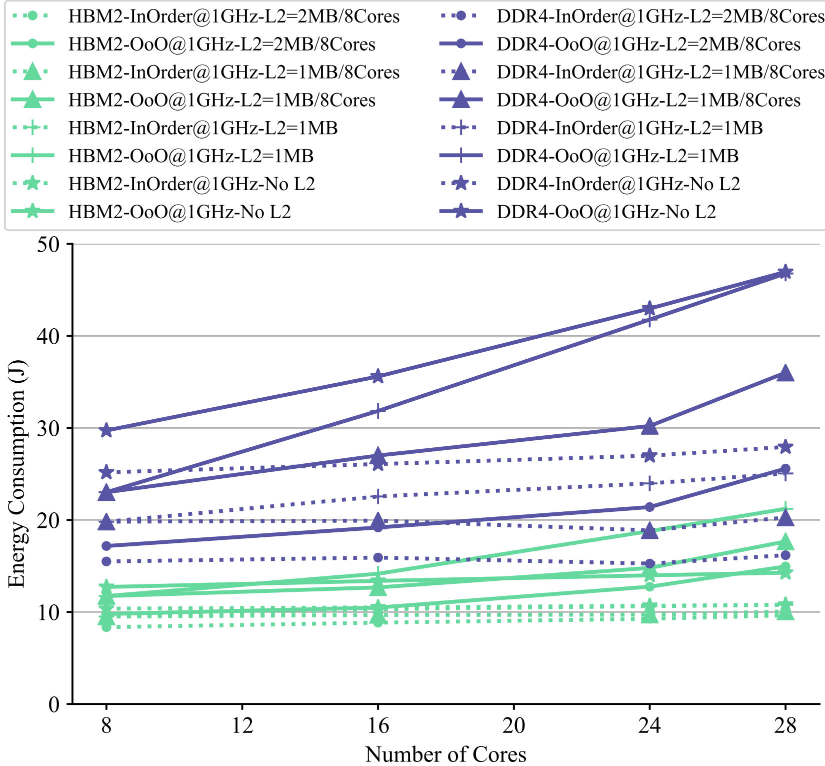


Figure 5.9: Energy scaling at 1GHz

performance for this KNL processor. We can observe that 16 ARM cores at 2GHz improves the performance of 32 Xeon Phi threads. In Figure 5.8, we observe that 28 ARM OoO cores at 1GHz match the performance of 32 Intel Xeon Phi KNL cores at 1.5 GHz. To have a fair comparison we simulate more ARM in-order and OoO systems with HBM2 at 1.5GHz, which is the operating frequency of KNL. The results of this comparison are depicted in Figure 5.10, and from there it can be seen that 16 OoO ARM cores match the performance of 32 KNL cores both at 1.5GHz. Figure 5.6 also shows that some 16, 24 and 28 ARM OoO cores at 2 GHz outperforms KNL performance significantly. Specifically, we found that 24 ARM OoO cores without L2 and with HBM2, match the performance of 32 KNL core and outperform KNL with just 28 cores.

However, when comparing the whole Bowtie2 sequence alignment application,

UNIVERSIDAD DE MÁLAGA

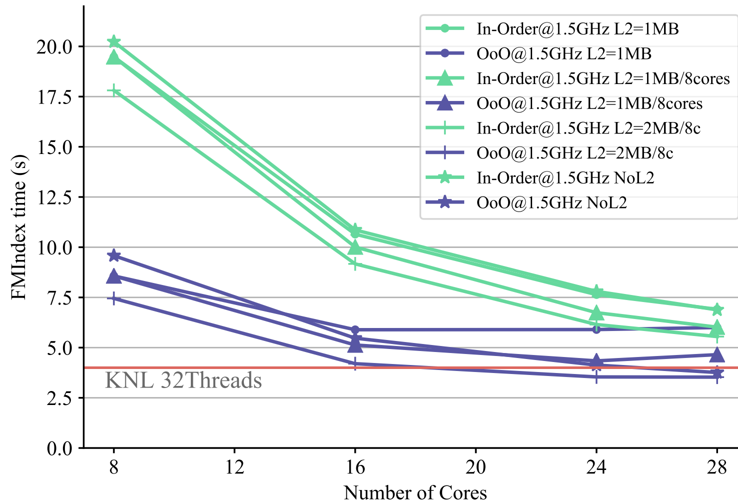


Figure 5.10: Performance scaling at 1.5GHz for HBM2

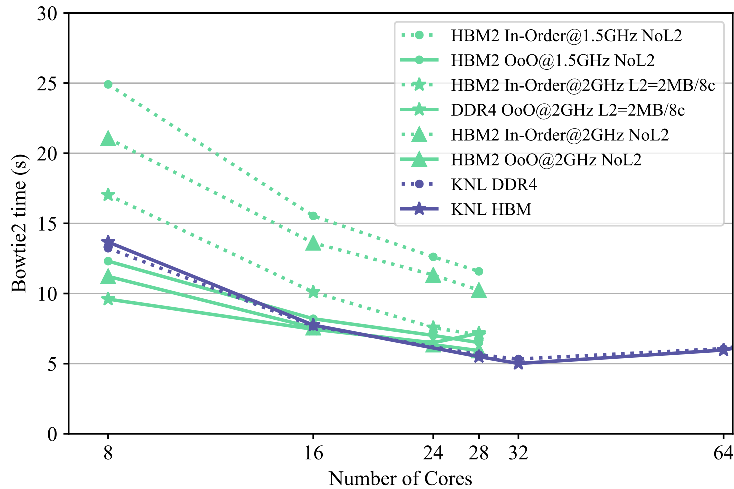


Figure 5.11: Bowtie2 execution times on Xeon Phi vs different ARM configurations

as shown in Figure 5.11, KNL performance is never surpassed by ARM cores (at least up to 28 cores). This can be explained because the no-FM-index part of the

UNIVERSIDAD DE MÁLAGA

Bowtie2 application is less memory bounded and is able to benefit more from the greater computing power from Intel Xeon Phi cores.

5.5 Conclusions

In this chapter, using the gem5-X architectural simulator, we explored architectures for optimizing performance and energy of the system for genome sequence alignment, using Bowtie2 for NGS which is memory bounded random access workload. Such memory bounded workloads do not require power hungry HPC compute nodes like Intel Xeon Phi KNL, but instead they require improvements in the memory bandwidth to enhance the overall performance and energy. In this work, we showed that by using high bandwidth memories like HBM2 along with energy efficient compute cores, for memory bounded NGS application (i.e Bowtie2), we can achieve up to 68% performance and 71% energy benefit as compared to a traditional system with DDR4. We also demonstrated that up to 47% energy savings can be achieved using many simple in-order cores, instead of fewer complex OoO cores at the same frequency. By scaling frequency, the number and type of cores, we achieved more than 8x energy savings. Lastly, we proposed a variety of architectures based on ARMv8 cores with HBM2 and demonstrated that 16 ARMv8 OoO cores with HBM2 outperforms 32-core Intel KNL processor.

UNIVERSIDAD DE MÁLAGA

6 Conclusions

In this thesis we have tackled the problem of the memory-processor bottleneck, also called *memory wall*, in memory bound applications. We have tried to reduce this bottleneck impact with different approaches.

Chapter 3 has focused on the application side, analyzing a well-known memory-bound application, as it is the FM-index based backward search application in the context of genome sequence alignment. We have performed an in-depth analysis of the throughput and behaviour of this algorithm on different computing systems, from conventional Intel Skylake Xeon processors (with DDR-type main memory technology) to the Intel Xeon Phi KNL processor, that integrates high-bandwidth memory technology (MCDRAM). Some solutions, previously proposed in literature, improve memory footprint and data locality exploitation. However, their impact on the throughput (queries per second) is relatively small. Some of them, as the k-step strategy, increase the date cache block size necessary to perform a single query operation but reduce the number of operations. This is granted almost at no cost in some GPU systems, where the block size is greater than in a typical CPU system.

In contrast to these results, the optimization that we have proposed to the FM-index data structure (called, split bit-vector sampled FM-index) reduces the data block size accessed in a single query, making it able to fit in a single cache block, and thus reducing the amount of blocks needed to load from main memory (memory bandwidth) but at the cost of increasing slightly the memory footprint of the whole index.

We have also developed a new benchmark, called RANDOM, able to perform a specific number of random memory accesses, mimicking the FM-index back-

UNIVERSIDAD
DE MÁLAGA

ward search algorithm behaviour and other similar memory-bound applications. The goal of this benchmark is to understand better the performance of applications with different arithmetic intensity and memory bandwidth requirements for each system, making us able to understand which architecture achieves better throughput for this type of memory access patterns.

In chapter 3 we have also experimentally evaluated an exact search algorithm based on the proposed FM-index data structure using three multi-core processors. Our proposal outperforms by about 60%, 90% and 135% the best of previous implementations, optimized and executed in Broadwell, Skylake and KNL processors, respectively.

The best performance was obtained in the Intel Xeon Phi (KNL) architecture, mainly because of the high peak random access memory bandwidth. Our implementation is able to obtain a throughput of 12G LFM/s, being about 3x faster than previous GPU implementations and about 4.4x faster than the GPU version implemented in the NVIDIA NVBIO bioinformatics library executed on a Tesla Pascal P100.

Once we understood how this kind of applications works and where the main bottlenecks are exactly located (and therefore limiting the applications performance and throughput), we moved to the development of new computer architectures, using Near-Data-Processing (NDP) configurations in order to push the performance limits even further and minimize the energy consumption.

Following this path, in chapter 4 we described some new processing-in-memory computer architectures optimized for this kind of workloads. We observed that modern architectures including deep cache hierarchies are not very efficient for this type of random access patterns. Our architectures are based on including general purpose, energy efficient ARM-like cores into the logical layer of a 3D-stacked memory cube. These architectures have been implemented and evaluated using the ZSim architectural simulator together with the Ramulator DRAM simulator, in order to accurately simulate a 3D-stacked cube with in-order cores integrated in the logic layer. We also use McPAT in order to estimate the power consumption of each architecture.

We have compared and analyzed both conventional DDR-type and PIM configurations for both the FM-index search algorithm and different parameters of the RANDOM benchmark. Our results show a significant improvement in performance and specially in energy efficiency when using small in-order cores versus typical x86 out of order cores. In brief, our setups improve more than 3× the throughput of the RANDOM benchmark and the FM-index application and, in some cases, with an energy efficiency of up to 40× better when compared with

UNIVERSIDAD DE MÁLAGA

systems using typical DDR-type memory technologies and deep cache hierarchies.

Lastly, in chapter 5 we try to get a more realistic knowledge of the performance of this type of memory-bound applications. In order to achieve this goal, we analyze the behaviour of the Bowtie2 [64] application, a sequence aligner widely used, which includes FM-index backward search algorithm on PIM architectures based on High Bandwidth Memory (HBM) devices. For that purpose, we used the gem5-X full-system architectural simulator, able to simulate full computer systems including the operating system. We simulate different configurations on different values of both in-order and OoO cores, both with typical DDR4 memory and new HBM memory.

Using HBM, we can achieve up to 68% performance and 71% energy benefits as compared to a traditional system with DDR4 memory. We also demonstrated that up to 47% energy savings can be achieved using many simple in-order cores, instead of fewer complex OoO cores at the same frequency. By scaling frequency, the number of cores and type of cores, we achieved more than 8x energy savings.

In summary, we started analyzing a memory-bound application with low performance due to the memory-processor bottleneck. After that, we have worked on improving that performance and the energy efficiency of this kind of applications (specifically those with random memory accesses) from several points of view:

a) in chapter 3 we tackle it from the software point of view, reducing the amount of memory we move,

b) in chapter 4 we move to the hardware part, proposing new architectures for random accesses,

c) finally, chapter 5 keeps on the hardware track, carrying out many experiments by using a complete, widely used sequence alignment application (Bowtie2) and running the experiments on a well-known full-system architectural simulator.

UNIVERSIDAD DE MÁLAGA

Results of this thesis derived in the following publications:

Exact Alignment with FM-index on the Intel Xeon Phi Knights Landing Processor
Jose M. Herruzo, Sonia Gonzalez-Navarro, Pablo Ibañez, Victor Viñals, Jesus Alastruey and Oscar Plata
Workshop on Accelerator Architecture in Computational Biology and Bioinformatics (AACBB'18) (co-located with HPCA 2018), Vienna (Austria), February 2018.

Optimizing Large Data Structures with Unpredictable Access Patterns in the Intel KNL Processor
Jose M. Herruzo, Sonia Gonzalez-Navarro and Oscar Plata
20th Workshop on Compilers for Parallel Computing (CPC'18), Dublin (Ireland), April 2018.

Accelerating Sequence Alignments Based on FM-index Using the Intel KNL Processor.
Jose M. Herruzo, Sonia González, Pablo Ibáñez, Víctor Viñals, Jesús Alastruey-Benedé, and Óscar Plata
IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), December 2018.

Boosting Backward Search Throughput for FM-index Using a Compressed Encoding.
Jose M. Herruzo, Sonia González, Pablo Ibáñez, Víctor Viñals, Jesús Alastruey-Benedé, and Óscar Plata
Proceedings of the 2019 Data Compression Conference (DCC 2019), 26–29 March 2019, Snowbird, Utah, USA, pp. 577.

Aceleración de una Aplicación con Acceso Intensivo e Impredecible a los Datos en el Procesador Intel Xeon Phi KNL
Jose M. Herruzo, Sonia Gonzalez-Navarro, Pablo Ibañez, Victor Viñals, Jesus Alastruey and Oscar Plata

UNIVERSIDAD DE MÁLAGA

XXIX Jornadas de Paralelismo (part of Jornadas Sarteco), Teruel (Spain), September 2018.

Results from chapters 4 and 5 are currently in evaluation in an international journal and an international conference, respectively.

6.1 Future work

Regarding FM-index data structure and algorithms, some future developments could be to integrate the techniques implemented in some higher level alignment software, like Bowtie [65] or Bowtie2 [64]. Other improvements could be the modification of the data structure in order to support more than 4 different symbols. The modifications already made to the data structure would make this step quite easy, with the only cost of an increase in the index memory footprint.

Other approaches for future work could be analyzing the performance of random access applications like FM-index in new 3D memory architectures, such as the new Micron boards with HMC and FPGA included in the same silicon board. This board is supposed to be able to run efficient and high performance computing workloads seizing the high memory bandwidth provided by the Hybrid Memory Cube.

On the other hand, regarding the development of new NDP computer architectures in order to improve the performance and efficiency of this kind of applications, there are several lines of work which deserve to be researched in near future. One of the most promising is the development of new heterogeneous PIM architectures including small energy-efficient general purpose cores under the memory modules, able to work together with typical host processors. This type of architectures presents lots of new challenges which would need to be overcome, like the communication models between main and 'accelerators' cores and the model and strategies of parallel programming necessary to efficiently take advantage of the architecture. In this field, one of our future ideas is developing a big.LITTLE-like system where the LITTLE cores are just under the memory modules, resulting in a system with big high performance cores and small energy efficient and memory oriented cores.

Lastly, other possible future proposals could be the exploration of specific purpose PIM architectures. There are several works already published in literature in this field, but new architectures and applications could be analyzed and

UNIVERSIDAD DE MÁLAGA

optimized, using FPGAs or specific logic hardware.

UNIVERSIDAD
DE MÁLAGA

Apéndice A
Random Memory Access Benchmark

We have developed a benchmark, called RANDOM, able to perform memory load operations following a random pattern similar that exhibited by the FM-index based backward search algorithm. It is a very useful tool in order to have a more accurate value of memory bandwidth when issuing this kind of memory accesses, usually much less efficient than sequential accesses.

Figure A.1 and A.2 show the RANDOM benchmark algorithm, in block diagrams and in pseudocode, respectively. RANDOM uses C randomly generated linked lists with no access locality. An array of head pointers is update a number of times following the linked lists. After each pointer update, the next list element is prefetched. This way, if C is large enough, the latency of all memory accesses is hidden, as shown in Figure A.5.

This way, this benchmark mimics the memory pattern of the accesses to the different versions of the *SFM* structure (see chapter 3).

We have performed several bandwidth tests in different systems, using the maximum number of hardware threads supported by system processors and with different number of linked lists (C).

For C values beyond 6, the bandwidth reaches a peak and remains stable (see Fig. A.3). For lower values, bandwidth is under the peak value because the memory latency cannot be completely hidden by the prefetching operations.

The maximum bandwidth corresponds to KNL MCDRAM, able to provide about 219 GB/s. However, this value is much lower than the peak 400 GB/s

UNIVERSIDAD DE MÁLAGA

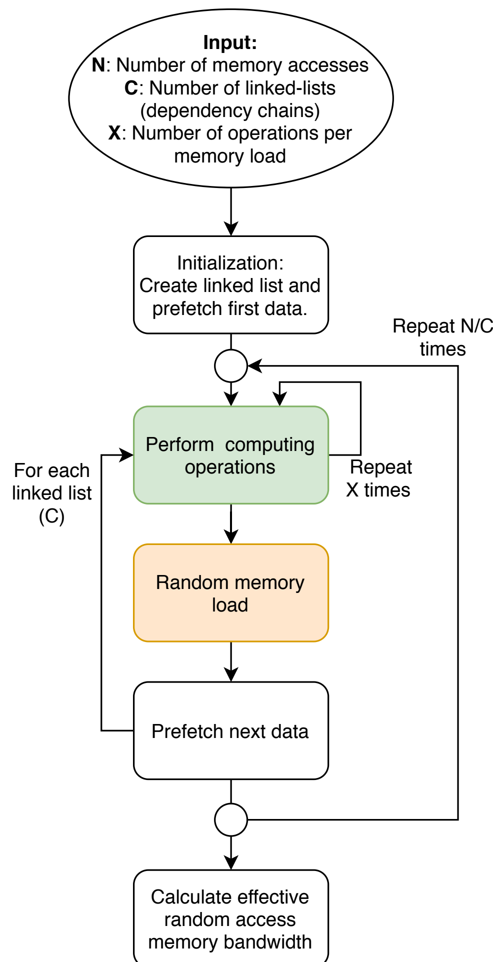


Figure A.1: RANDOM benchmark blocks diagram

reported for the STREAM benchmark [80]. On the other hand, the peak bandwidth provided by the DDR4 DRAM memory is about 68 GB/s (Skylake) and 44 GB/s (Broadwell).

The memory access latencies were also evaluated using the RANDOM benchmark. Three load tests were used: *low load*, *medium load* and *high load*:

- In the low load test, only one hardware thread in the complete processor

UNIVERSIDAD DE MÁLAGA

Benchmark: RANDOM

Input: *N*: Number of random memory accesses
C: Number of linked lists (dependency-chains)
p: Array of head pointers to the linked lists

```
   begin
1: for i from N-1 to 1 step C
2:   for k from 0 to C step 1
3:     p[k] = p[k]→next
4:     prefetch(p[k])
5:   end for
6: end for
   end
```

Figure A.2: RANDOM benchmark

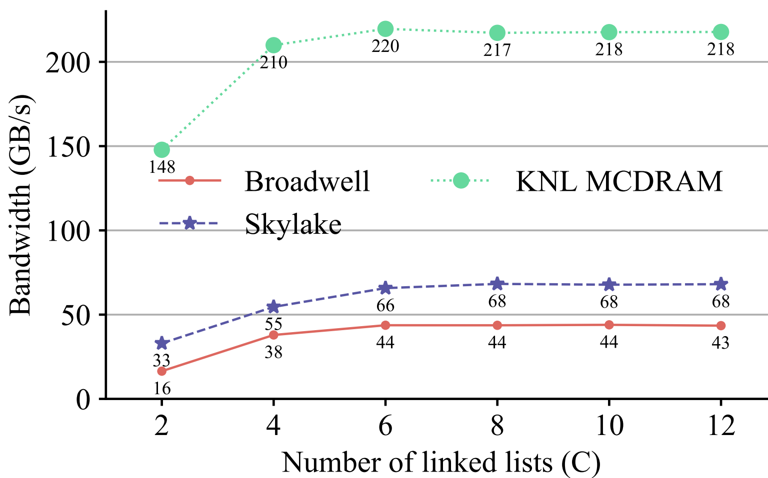


Figure A.3: Memory bandwidth for Broadwell, Skylake and KNL obtained with the RANDOM benchmark for various values of C

executes the benchmark that runs over a single linked list (C=1). All the others threads remain idle.

UNIVERSIDAD DE MÁLAGA

- In the medium load test, the maximum number of hardware threads in each processor runs over a single linked list.

- In the high load test, the maximum number of hardware threads (same as medium load) execute the benchmark.

In the three load configurations, every thread, except one, runs over several linked list in order to have a high load in the system. The remaining thread runs just over one linked list in order to accurately measure the memory access latency. The number of linked lists was selected to be the one that achieves the best memory bandwidth (see Figure A.3).

Figure A.4 shows the RANDOM latency results for the three processor architectures. It can be noted that the latency increases significantly with the load in the system.

This behaviour is expected as, when the amount of simultaneous queries increases, accesses to hardware shared resources are much more likely to conflict. Some of these shared resources could be the memory channels, memory banks or the core interconnection network.

Additionally, this benchmark has been expanded in order to perform a number of arithmetic operations per memory access, as shown on green highlighted block in Figure A.1. This allows us to easily change the arithmetic intensity of the software, which is very useful, specially, for creating and evaluating roofline models for each system in different conditions and for evaluating the different applications and computing/memory operations ratio that fits better for a specific

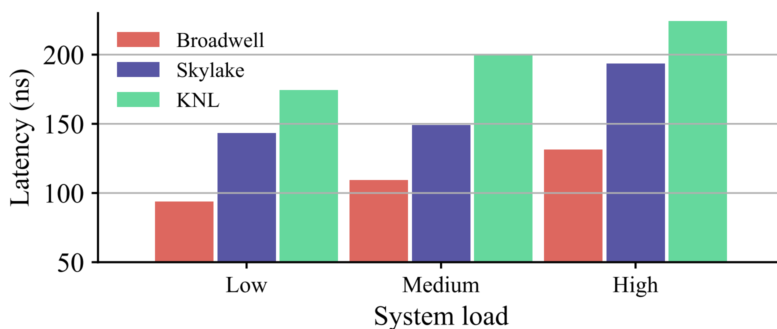


Figure A.4: Comparison of RANDOM memory latencies

UNIVERSIDAD DE MÁLAGA

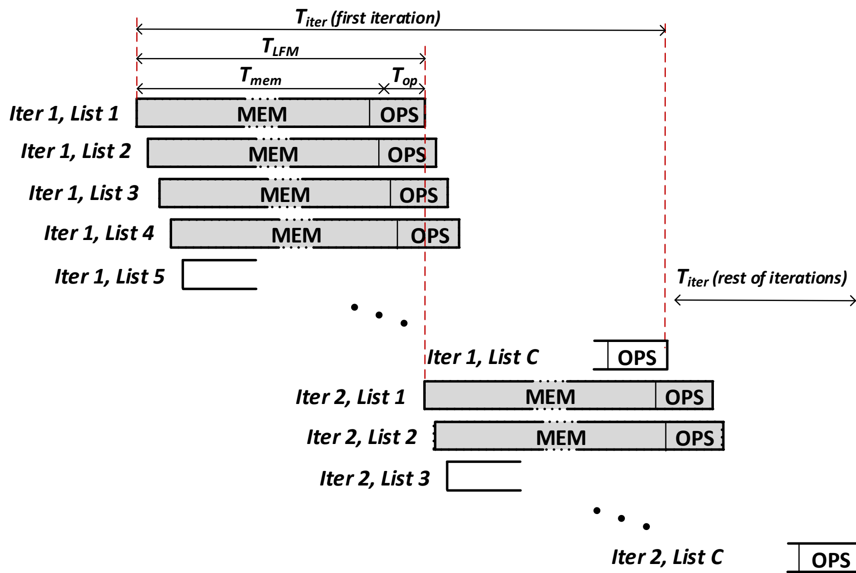


Figure A.5: RANDOM benchmark timing model

system. This utility has been extensively used in chapter 4.

RANDOM benchmark can be configured for all the relevant parameters which can characterize a specific memory access pattern:

- Data structure size.
- Number of threads.
- Number of parallel linked list.
- Arithmetic intensity.
- Datatype for arithmetic operations.
- Size of cache block.
- Number of cache blocks to be loaded simultaneously.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Apéndice B
Resumen en español

La aparición de aplicaciones con un uso intensivo de datos ha despertado un gran interés en el diseño de técnicas para procesamiento eficiente y escalable. Desde el punto de vista del almacenamiento, el rendimiento aumenta si los datos están organizados en la memoria principal de forma que los patrones de acceso puedan explotar la localidad de los datos. Sin embargo, cuando aparecen patrones de acceso impredecibles, las jerarquías de caché que se encuentran en los procesadores modernos funcionan ineficientemente y con frecuencia causan una gran demanda de ancho de banda de memoria. Un ejemplo de aplicaciones que presentan este comportamiento son algunas usadas en bioinformática para el alineamiento de secuencias de ADN, como por ejemplo aquellas basadas en el algoritmo y estructura FM-index.

Además, el reducido ritmo de mejora del rendimiento del sistema de memoria, con respecto a los sistemas de cómputo de los últimos años, ha llevado a la aparición de un cuello de botella en el transporte de los datos desde donde se almacenan hasta donde van a ser procesados. Esto se ha llamado en la literatura 'memory-wall' [113].

En los últimos años, se han comenzado a fabricar memorias DRAM con varias capas apiladas en 3D. Por ejemplo, Hybrid Memory Cube (HMC) [81] (Micron Technology) y High Bandwidth Memory (HBM) [4] (AMD y Hynix) proporcionan anchos de banda de varios cientos de GB/s. Estas tecnologías se están incluyendo en diferentes procesadores de alto rendimiento, como GPUs de AMD (Radeon R9 Fury) y NVIDIA (Pascal), y CMPs de Intel (Xeon Phi Knights Landing).

UNIVERSIDAD DE MÁLAGA

B.1 Motivación y Temas de Investigación

La combinación de la aparición del mencionado cuello de botella y la creciente importancia de las aplicaciones de procesamiento intensivo de datos, muy limitadas por el sistema de memoria, crea un importante problema que debe ser abordado. Por ello, en esta tesis nos proponemos afrontar este problema e intentar reducir su efecto en la medida de lo posible.

El principal objetivo de esta tesis es el diseño de nuevas soluciones arquitecturales y algorítmicas para superar el problema del cuello de botella conocido como memory-wall y mejorar el rendimiento de aplicaciones con gran uso de memoria que no son capaces de beneficiarse lo suficiente de las jerarquías de memoria actuales. Además, creemos que actualmente y especialmente en el futuro, es esencial centrarse en la eficiencia energética, un factor cuya importancia crece cada día y uno de los factores más limitantes en la computación de alto rendimiento.

Las principales contribuciones de esta tesis son:

- Analizamos el comportamiento de aplicaciones con accesos de memoria aleatorios, que no aprovechan correctamente las nuevas arquitecturas de memoria con jerarquías cache profundas. Específicamente, analizamos la estructura de datos FM-index y un algoritmo de búsqueda de secuencias basado en esa estructura, ampliamente usado en el alineamiento de secuencias en el genoma.
- Después de este análisis y de obtener un conocimiento más detallado del cuello de botella de la memoria, proponemos una nueva versión de FM-index que permite reducir el consumo de ancho de banda de memoria, de forma que mejora significativamente el rendimiento computacional.
- Proponemos una nueva arquitectura energéticamente eficiente, basada en un cubo de memoria en 3D (3D-Stacked) al que añadimos unos núcleos sencillos de bajo consumo en su capa lógica. Esta arquitectura permite la ejecución cerca de los datos (near-data-processing)
- También realizamos un estudio experimental de varias arquitecturas con diferentes tecnologías de memoria (DDR y HBM) y núcleos de procesamiento de distintos tipos, explotando, en algunos casos, procesamiento en la memoria (PIM). La aplicación de referencia es Bowtie2, una aplicación completa para el alineamiento de secuencias en el genoma. La implementación y evaluación de estas arquitecturas se realiza utilizando un simulador arquitectural basado en gem5.

UNIVERSIDAD DE MÁLAGA

Esta tesis se organiza de la siguiente forma: en el capítulo 1 incluímos nuestras motivaciones para llevar a cabo esta tesis y una breve descripción de arquitecturas centradas en los datos y aplicaciones intensivas en memoria. El capítulo 2 incluye una descripción de conceptos y trabajos relacionados a partir de los cuales se desarrolla esta tesis. En el capítulo 3 analizamos la búsqueda exacta y las estructuras de datos de FM-index, asi como sus variaciones presentes en la literatura, y proponemos una nueva versión de estos algoritmos. Los capítulos 4 y 5 presentan nuevas arquitecturas orientadas a aumentar el rendimiento de la memoria. Finalmente, el capítulo 6 presenta las concluiones de esta tesis. En este apéndice hemos hecho un resumen a partir del capítulo 3 (es decir, hemos resumido los capítulos más importantes de la tesis), pero intengrando los conceptos que se introducen en el capítulo 2 a lo largo de todo el resumen.

B.2 Alineamiento de Secuencias con FM-index

FM-index es un índice de texto basado en la transformada de Burrows-Wheeler [21] que permite búsquedas rápidas de cadenas en textos de referencia que ocupan un espacio de memoria reducido. La transformada de Burrows-Wheeler (BWT) es una permutación de una cadena de caracteres.

FM-index combina compresión e indexación de tal forma que al realizar una consulta no se pierde rendimiento comparado con una consulta realizada sobre un índice de texto completo. FM-index está compuesto por dos estructuras de datos derivadas de la transformada de Burrows Wheeler (BWT): el vector C y la matriz Occ.

El vector C almacena en $C[c]$ el número de ocurrencias en la transformada BWT de los símbolos lexicográficos menores que c. Por otro lado, $Occ[c, i]$ contiene el número de ocurrencias del símbolo c en el prefijo $\{1..i\}$ de la transformada BWT, siendo $1 \leq i \leq n + 1$.

B.2.1 Análisis de FM-index

El algoritmo de búsqueda exacta desarrollado en [33], también llamado *backward search* (BS), puede realizar la búsqueda con una complejidad de $\Theta(p)$. Este algoritmo se muestra en la Figura B.1.

La operación principal en el algoritmo de búsqueda se denota *Last-To-First Mapping* (LFM), y se realiza llamando a la función $LF()$, definida de la siguiente

UNIVERSIDAD DE MÁLAGA

```
Algorithm BS: Backward Search Based on FM-index
Input: FM-index of T text (C & Occ), Q query, n:|T|, p:|Q|
Ouput: (sp,ep): Interval pointers of Q in T
   begin
1: sp = C[Q{p}]; ep = C[Q{p}+1]
2: for i from p-1 to 1 step -1
3:    sp = LF(Q{i},sp); ep = LF(Q{i},ep)   2 LFM-chains
4: end for
5: return (sp+1,ep)
   end
```

Figure B.1: Algoritmo básico de búsqueda *BS* basado en FM-index.

manera:

$$LF(Q\{i\}, u) = C[Q\{i\}] + Occ[Q\{i\}, u], \tag{B.1}$$

donde i es el índice del bucle y u es *sp* o *ep*.

Cada iteración del bucle 2−4 en el algoritmo BS accede a la cadena Q y hace dos llamadas a la función $LF()$, una con *sp* y la otra con *ep*. Cada puntero (*sp* o *ep*) se actualiza utilizando su valor calculado en la iteración previa. Por tanto, se tienen dos cadenas de dependencias, una para *sp* y la otra para *ep*. Denotamos estas cadenas como *LFM-chains*.

B.2.1.1 Patrón de Accesos a Memoria

Una de las principales limitaciones de rendimiento del algoritmo BS está relacionada con la comunicación entre el procesador y la memoria. Al ejecutar este algoritmo en un procesador fuera de orden, se emiten dos cadenas LFM para cada consulta de búsqueda, superponiendo su ejecución. Cada LFM accede a un elemento de la matriz C y a otro de la matriz *Occ*. Sin embargo, estos dos accesos interactúan con la jerarquía de memoria de forma muy diferente. Probablemente, la matriz C completa estará almacenada en la caché L1, dado su pequeño tamaño (ya que el número de símbolos X del alfabeto es pequeño en muchos dominios de aplicación). La matriz *Occ*, por el contrario, es una estructura mucho más grande y con un patrón de acceso a memoria no predecible: sus accesos están distribuidos por toda la estructura, y no muestran ninguna localidad espacial ni temporal. Por lo tanto, este tipo de accesos causan una alta tasa de fallos en la jerarquía cache.

UNIVERSIDAD DE MÁLAGA

Sea α el promedio de bloques cache leídos desde la memoria principal por cada par de LFMs en la misma iteración del bucle en el algoritmo BS. Se puede escribir, $\alpha = 1 + (1 - r)$, siendo r la probabilidad de que *sp* y *ep* apunten a dos valores almacenados en el mismo bloque cache.

El concepto de *intensidad operacional* o *intensidad aritmética* se usa para correlacionar las operaciones (cálculos) con el acceso a la memoria [110]. Esta métrica es la relación entre el número de operaciones (trabajo) y la cantidad de tráfico de datos (en bytes) que provoca. En el caso del algoritmo BS basado en FM-index, utilizamos el número de LFMs ejecutados por byte transferido. En consecuencia, llamamos a esta métrica *intensidad de búsqueda* o *Search Intensity* (*SI*). En promedio, un par de LFMs necesita leer α bloques cache de la memoria principal. Por lo tanto, la *SI* del algoritmo BS para un bloque cache de tamaño B en bytes es $SI = 2/(\alpha \times B)$ LFMs/byte.

B.2.1.2 Rendimiento

La ejecución de un par de cadenas LFM emitidas en una consulta de búsqueda se puede modelar con dos fases lógicas (ver Figura B.2 (izquierda)). La primera fase, *MEM*, corresponde a las operaciones de memoria asociadas a una LFM. La latencia de esta fase está determinada principalmente por el acceso a *Occ*. La segunda fase, *OP*, corresponde a las operaciones de computación en una LFM. Básicamente, esta fase comprende la ejecución de tres instrucciones de acceso a memoria y una suma (ver expresión (B.1)).

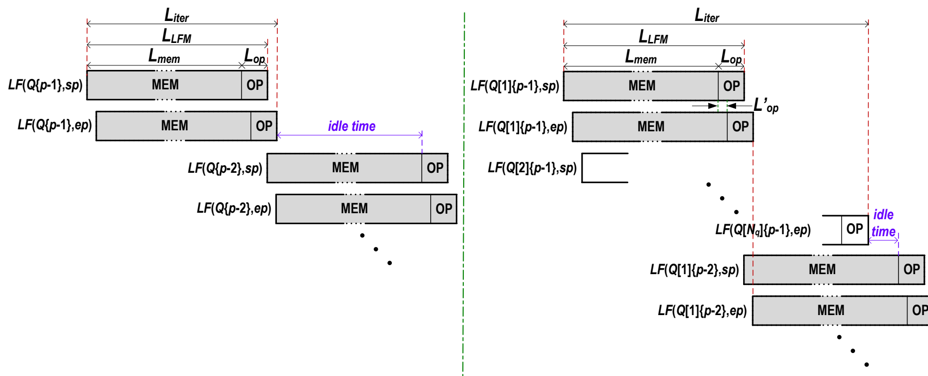


Figure B.2: Modelos temporales para algoritmos BS (izquierda) y OBS (derecha), donde L_x representa latencias.

UNIVERSIDAD DE MÁLAGA

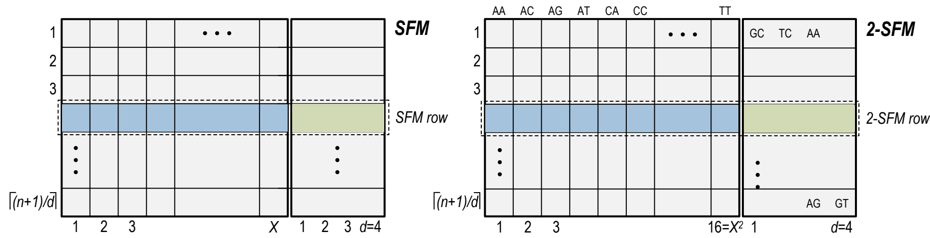


Figure B.3: Las estructuras de datos de *Sampled FM-index* (izquierda) y *k-step Sampled FM-index* (derecha), considerando $d = 4$, $k = 2$ y el genoma humano ($X = 4$).

B.2.1.3 K-Step Sampled FM-index

El tamaño de Occ es enorme para textos grandes (T). Para reducir el tamaño de la estructura de datos en memoria, Occ puede reemplazarse con otra matriz más pequeña, $rOcc$, que almacena una de cada d columnas de Occ [33]. Cuando el algoritmo de búsqueda requiere algunos datos de Occ que no están almacenados en $rOcc$, estos son recalculados usando $rOcc$ y el trozo (*bucket*) de BWT de T asociado y que denotamos como $bBWT$. Combinando estas últimas estructuras obtemos una nueva que denotamos como *SFM*. Estas estructuras de datos se pueden ver en la Figura B.3 (izquierda).

El algoritmo de búsqueda exacta basado en *SFM* es similar al algoritmo BS, pero tiene un coste computacional mayor para cada operación LFM debido a las reconstrucciones de Occ.

La localidad de los datos puede mejorarse aún más si se consultan varios símbolos de Q en el cálculo de una LFM, como se propone en [23]. Para consultar k símbolos, se reemplaza el alfabeto original Σ por el conjunto de k-tuplas cuyas entradas se obtienen de Σ (permutaciones con repetición). Este cambio en el alfabeto requiere modificar la estructura de datos de FM-index, que denotamos como *k-step sampled FM-index* (*k-SFM*). Esto se muestra en la Figura B.3 (derecha).

La versión del algoritmo de búsqueda para una estructura FM-index *k-step* se adapta bien en algunas arquitecturas, como GPUs, pero resulta en un gran incremento del tamaño de las estructuras de datos en memoria haciéndolo inviable para valores de k mayores de 2.

UNIVERSIDAD DE MÁLAGA

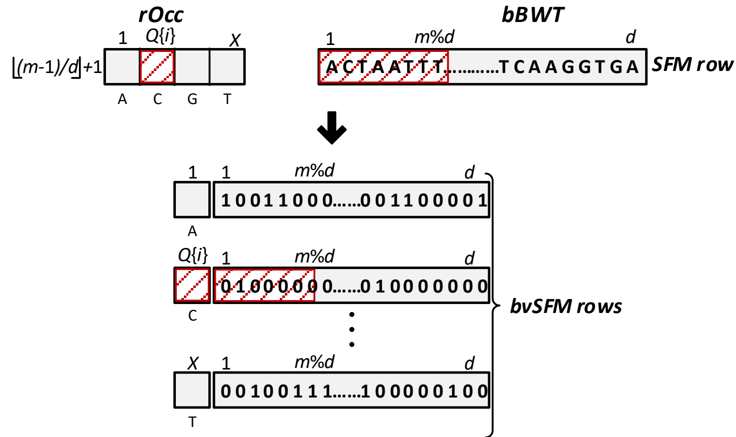


Figure B.4: Estructura de datos *SFM* original (arriba) y nueva estructura de datos *bvSFM* (abajo). Los datos a los que se accede durante el cálculo de una LFM están marcados en rojo.

B.2.2 Split Bit-Vector k-Step Sampled FM-index

La búsqueda utilizando *k-step sampled FM-index* tiene un impacto limitado en la intensidad de búsqueda (SI). El aumento en el número de LFMs por consulta es compensado por un aumento en α, la cantidad promedio de bloques cache accedidos, ya que las filas de *k-SFM* son más grandes que las filas de *SFM*, sobrepasando el tamaño típico de un bloque cache. Con el fin de aumentar la intensidad de búsqueda para mejorar el rendimiento, se debe reducir el valor de α. La Figura B.4 (arriba) muestra una fila de la estructura de datos *SFM*. Todas las entradas a las que se accede en el cálculo de una LFM están marcadas en rojo. Se puede apreciar que se accede a una única entrada en $rOcc$, sin embargo, si X es lo suficientemente grande, la subcadena accedida en $bBWT$ estará almacenada en un bloque cache diferente.

Para reducir los fallos cache, proponemos reorganizar el diseño *k-SFM* y cambiar la codificación de datos de tal forma que todos los datos necesarios para calcular una LFM se almacenan en un número mínimo de bloques cache. La nueva estructura de datos, denotada por *k-bvSFM*, se llama *split bit-vector k-step sampled-FM-index*. Nuestra solución proviene de la observación de que solo se lee una de las entradas de X^k k-$rOcc$ para cada cálculo de LFM (ver Figura B.4 (arriba) para $k = 1$).

UNIVERSIDAD DE MÁLAGA

La estructura *k-bvSFM* se obtiene de *k-SFM* mediante dos transformaciones. En primer lugar, cada fila *k-SFM* se divide en X^k filas, donde cada una de ellas incluye una sola entrada *k-rOcc* con el *bucket* completo. En segundo lugar, cada *bucket* se codifica como un mapa de bits donde cada símbolo se representa con un bit. Esto se lleva a cabo de la siguiente forma: dada una fila en *k-bvSFM* que corresponde a la entrada t en la fila *k-rOcc* original, su *bucket* asociado se codifica como un mapa de bits de longitud d, donde un símbolo en el *bucket* está representado por 1 si es igual al asociado a dicha entrada t, o por 0 en caso de que sea diferente.

Gracias a esta transformación, la nueva función de recuento de símbolos necesaria para reconstruir *Occ* a partir de *k-rOcc* se reduce a una operación *population count*, es decir, a contar la cantidad de bits con valor 1 en el *bucket* asociado a un símbolo dado $Q\{i\}$. Esta nueva estructura de datos almacena de forma compacta todos los datos necesarios para calcular una LFM, lo que minimiza la cantidad de bloques cache a los que se accede en cada operación. La Figura B.4 (abajo) muestra, por ejemplo, las X^k filas de *k-bvSFM* para $k = 1$.

Sin embargo, estos beneficios tienen el coste de una mayor ocupación de memoria, ya que cada entrada de *k-SFM* (de tamaño $\log_2(X^k) \times d$ bits) es reemplazada por X^k *bitmaps* (de tamaño $X^k \times d$ bits) .

El diseño *k-bvSFM* reduce el número de bloques cache necesarios para el cálculo de una LFM. Por lo tanto, el valor de α se reduce, lo que resulta en un aumento de la intensidad de búsqueda en comparación con versiones anteriores del algoritmo.

La Tabla B.1 muestra las propiedades computacionales de los algoritmos de búsqueda basados en diferentes versiones de FM-index. Todas las versiones incluyen el solapamiento de búsquedas para ocultar la latencia. Los cálculos se han realizado suponiendo un alfabeto de cuatro caracteres, debido a que la aplicación

Table B.1: Propiedades del algoritmo de búsqueda para diferentes versiones de FM-index (p.ej. $k1\text{-}32SFM$ corresponde a *k-SFM* con $k = 1$ y $d = 32$).

Versión	Tamaño fila (bloq cache)	α (bl. cache)	SI (LFMs/B)	Instrucciones por LFM
k1-32SFM	1	1.08	0.0288	33
k1-192SFM	1	1.07	0.0293	77.5
k2-16SFM	2	2.28	0.0274	37.5
k2-128SFM	2	2.15	0.0290	98.5
k2-64bvSFM	1	1.088	0.0574	23.5
k2-96bvSFM	1	1.081	0.0578	38

UNIVERSIDAD DE MÁLAGA

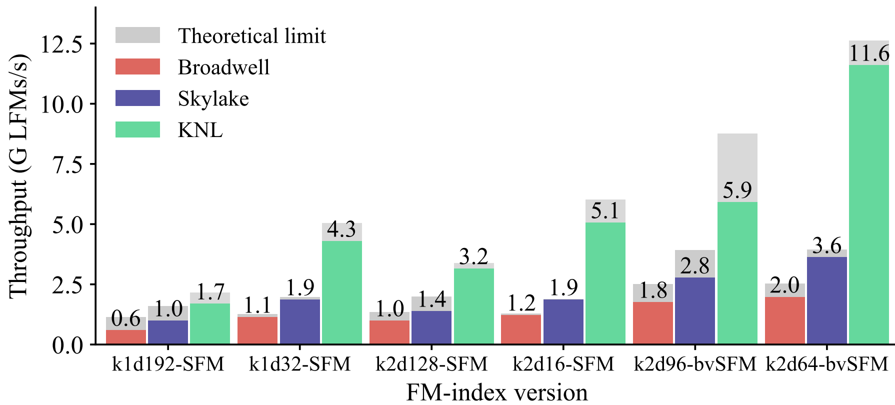


Figure B.5: Límites teóricos y rendimiento para diferentes versiones de FM-index.

seleccionada para evaluar nuestras propuestas busca cadenas en una secuencia de ADN, formado por cuatro caracteres diferentes.

B.2.3 Evaluación Experimental

La Figura B.5 muestra los límites teóricos y los resultados experimentales para diferentes versiones de FM-index evaluadas en tres procesadores distintos (Broadwell, Skylake y KNL). En general, los valores experimentales se aproximan de forma razonable a los valores teóricos esperados. El rendimiento real es el 95% del límite teórico para las versiones del algoritmo con buckets de menos de 64 bits y en torno al 80% para las versiones con buckets más grandes. Esto se debe a que el conteo de coincidencias en buckets de más de 64 bits provoca más fallos del predictor de saltos (branch predictor).

Podemos observar como las versiones del algoritmo basados en nuestra propuesta consiguen una mejora de rendimiento del 60 y el 90% en Broadwell y Skylake respectivamente. En KNL, nuestra propuesta mejora el rendimiento de las soluciones previas por un 135%. Además, el rendimiento en KNL de la mejor versión (k2d64-bvSFM) es 6 y 3 veces mejor que el conseguido en Broadwell y Skylake, respectivamente.

Finalmente, las Figuras B.6 y B.7 muestran el modelo *roofline* [110] para los algoritmos de búsqueda basados en diferentes versiones de FM-index. Este

UNIVERSIDAD DE MÁLAGA

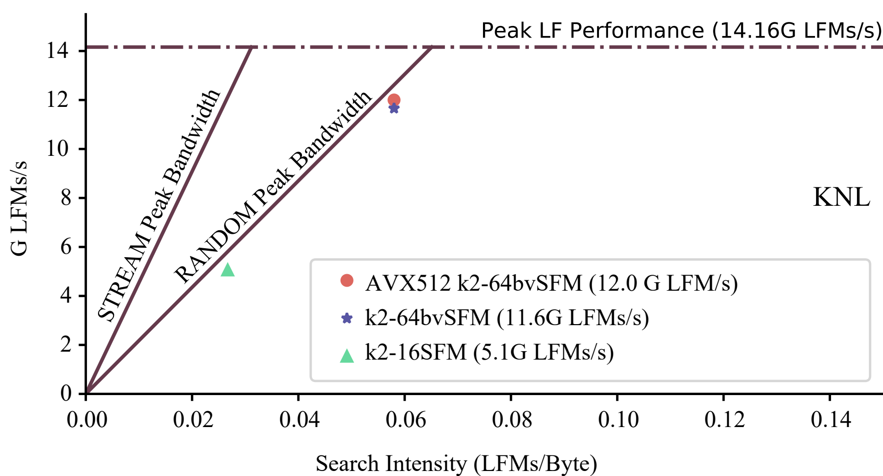


Figure B.6: Modelo *roofline* para Knights Landing (KNL).

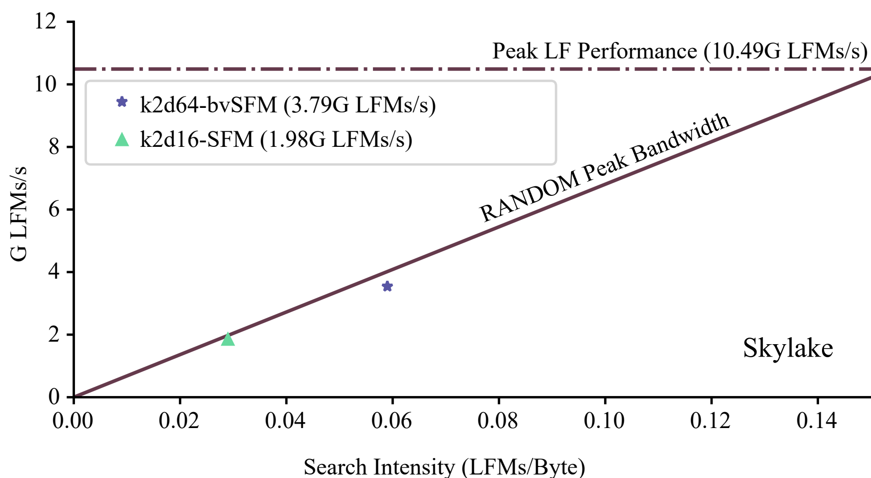


Figure B.7: Modelo *roofline* para Skylake (SKL).

modelo considera dos límites diferentes de ancho de banda:

- Ancho de banda máximo de la memoria principal.

UNIVERSIDAD DE MÁLAGA

- Ancho de banda para accesos aleatorios.

Como puede observarse en las figuras, este segundo límite es el factor limitante de la mejor versión (k2d64-bvSFM) para todas las arquitecturas estudiadas.

B.3 FM-index y Procesado en Memoria

Las aplicaciones con patrones de acceso a memoria aleatorios e impredecibles, como el algoritmo de alineamiento de secuencias basado en FM-index descrito en la sección anterior, no alcanzan beun rendimiento en las arquitecturas de procesamiento tradicionales, con una jerarquía de varios niveles de memoria cache. La mayoría de estas aplicaciones no obtienen ninguna ventaja de las caches o de la precarga de datos hardware, siendo en ocasiones incluso penalizadas por el aumento de latencia en los accesos de memoria.

Un ejemplo de este tipo de aplicaciones es el FM-index, ya explicado en apartados anteriores. Los accesos de memoria impredecibles y aleatorios, junto con el poco cómputo requerido, hacen de esta aplicación una buena candidata para ser implementada en arquitecturas de Procesamiento Cercano a los Datos y de Procesamiento en Memoria (NPD y PIM respectivamente por sus siglas en inglés).

B.3.1 Diseño de Arquitecturas y Simulación

Se han establecido tres arquitecturas de sistemas diferentes, con el objetivo de evaluar el rendimiento de los accesos a memoria aleatorios en una arquitectura de procesamiento en memoria (PIM).

Tal y como se muestra en la Tabla B.2, comparamos dos arquitecturas con tecnologías de memoria DDR y con 64 y 36 núcleos fuera de orden a diferentes frecuencias, con una configuración PIM basada en tecnología de memoria 3D (tipo HMC o HBM) y con 64 núcleos en orden, muy energéticamente eficientes a una baja frecuencia. Esta configuración PIM se muestra en la Figura B.8.

B.3.2 Estimación de Area y Consumo Energético

Para obtener una estimación del consumo energético y área utilizada por las diferentes configuraciones hemos usado tanto la herramienta McPAT como una estimación basada en procesadores reales.

UNIVERSIDAD DE MÁLAGA

Table B.2: Arquitecuras hardware simuladas

	DDR Setup 1	DDR Setup 2	PIM	Intel i7-8700
Núcleos	64 @ 2.4 GHz	36 @ 3.6 GHz	64 @ 1.5 GHz	6 @ 3.2-4.6GHz
T. Núcleo	OoO	OoO	in-Order	OoO
Hilos HW	1	1	1	2
Arquitectura	x86	x86	ARM-Like*	x86
Can. Memoria	4	4	-	2
Freq. Memoria	1600/2400 DDR3/DDR4	1600/2400 DDR3/DDR4	2500 3D-stacked	2400 DDR4
Bloq. Cache	64B	64B	32/64B	64B
Tecnología	22 nm	22 nm	28 nm	14 nm
L1 Cache (L1D/L1I)	32K/32K 3 cycles Latency	32K/32K 3 cycles Latency	8K/8K 3 cycles latency	32K/32K 8-way set associative
L2 Cache	256K 10 cycles Latency 8-way set assoc.	256K 10 cycles Latency 8-way set assoc.	-	256K 4-way set associative.
L3 Cache	16M Shared 30 cycles Latency 16-way set assoc. 6 banks H3 Hash	16M Shared 30 cycles Latency 16-way set assoc. 6 banks H3 Hash	- -	2M 16-way set associative

*We simulate In-order cores with a similar performance to ARM cores.

Según McPAT, cada núcleo PIM a 1500MHz usa un área de en torno a 2 mm^2 y consume en torno a 0.5 W por núcleo usando una tecnologías de 28 nm y 0.61 mm^2 de área y 0.20 W por núcleo usando tecnología de 22 nm.

Además, estos núcleos son similares a los núcleos A35 más pequeños presentados por ARM, pero a frecuencias más altas. Basándonos en esto, estimamos un área de 0.4 mm^2 por cada núcleo pequeño y un consumo de aproximadamente

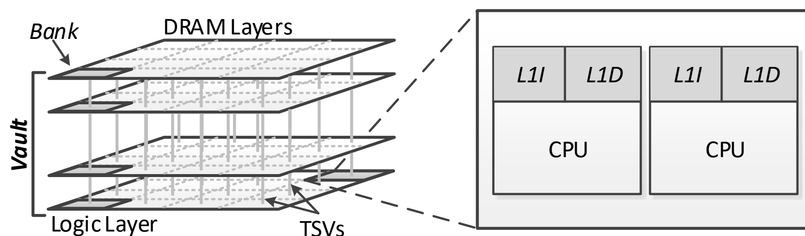


Figure B.8: Diagrama de arquitectura PIM

UNIVERSIDAD DE MÁLAGA

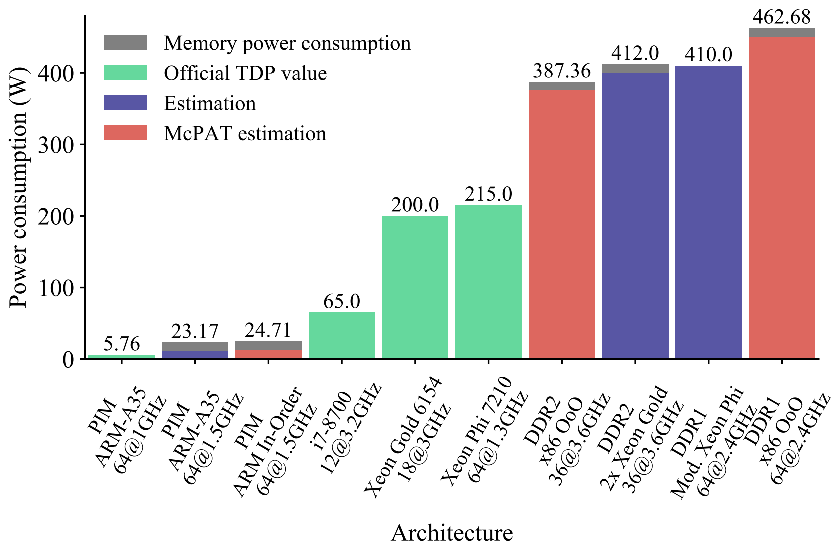


Figure B.9: Consumo de energía por procesador

180 mW.

Por otro lado, nuestras estimaciones de consumo para los procesadores de alto rendimiento varían en torno a 370 W y 450 W, datos mucho más altos que las configuraciones de bajo consumo.

Los datos de consumo se muestran en la Figura B.9.

B.3.3 Evaluación Experimental

Hemos realizado pruebas en una máquina real y en 4 arquitecturas simuladas diferentes, incluyendo variaciones en el tipo de memoria, el número de núcleos y el tipo de estos. Todas las arquitecturas se muestran en la Tabla B.2.

Estas pruebas han sido llevadas a cabo usando el simulador ZSim [97] en combinación con Ramulator-PIM [61]. La versión pública de ZSim ha sido modificada para que soporte la comunicación directa entre ZSim y Ramulator, simplificando la ejecución de los experimentos.

Hemos realizado varios experimentos usando los benchmarks STREAM [80] y RANDOM (Apéndice A). La Figura B.10 muestra los resultados de la ejecución

UNIVERSIDAD DE MÁLAGA

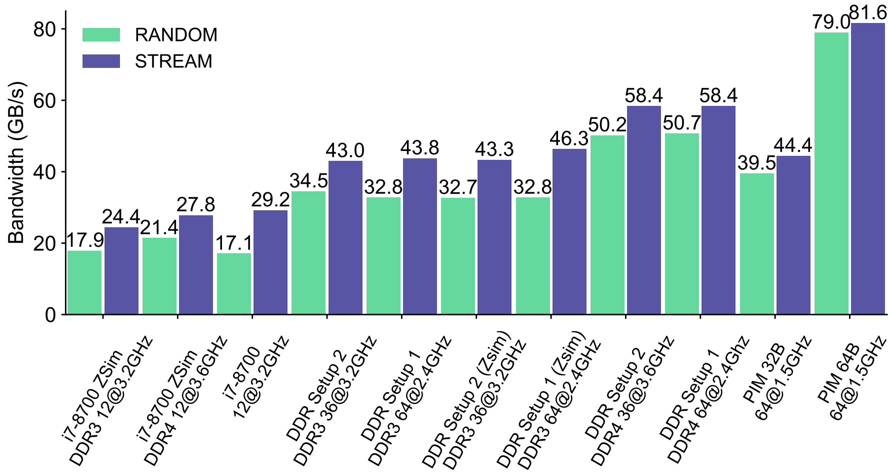


Figure B.10: Benchmark STREAM y RANDOM para distintas arquitecturas

de estos benchmarks en las distintas arquitecturas. Podemos observar claramente como el rendimiento para las arquitecturas PIM es mucho mayor, consiguiendo un incremento en el ancho de banda entre 1.4 y 3.4 veces al compararlo con arquitecturas tradicionales. Con respecto a RANDOM, podemos observar que se acerca mucho más a los resultados de STREAM para las arquitecturas PIM. Esto es debido a la reducida latencia y jerarquía cache presentes en este tipo de arquitecturas.

Además, para comprobar el funcionamiento de una aplicación real, con un patrón de accesos a memoria aleatorio hemos utilizado la aplicación FM-index, comparando tres diferentes versiones del algorithmo. Esto se puede observar en la Figura B.11. En este caso, las arquitecturas PIM consiguen hasta 3.7 veces mejor rendimiento que las arquitecturas tradicionales, siendo la versión k2d64bv la que obtiene mejores resultados. También podemos observar como las arquitecturas PIM con un bloque cache de 32 bytes obtiene resultados muy parecidos a la que usa bloques cache de 64 bytes. Esto se debe a que la aplicación analizada no utiliza más de 32 bytes de cada bloque cache.

En la Figura B.12 mostramos también el modelo roofline para la configuración PIM. En esta figura podemos ver como los resultados obtenidos con el benchmark RANDOM se adaptan muy bien al modelo roofline, acercándose mucho al ancho de banda límite para valores pequeños de la intensidad aritmética.

UNIVERSIDAD DE MÁLAGA

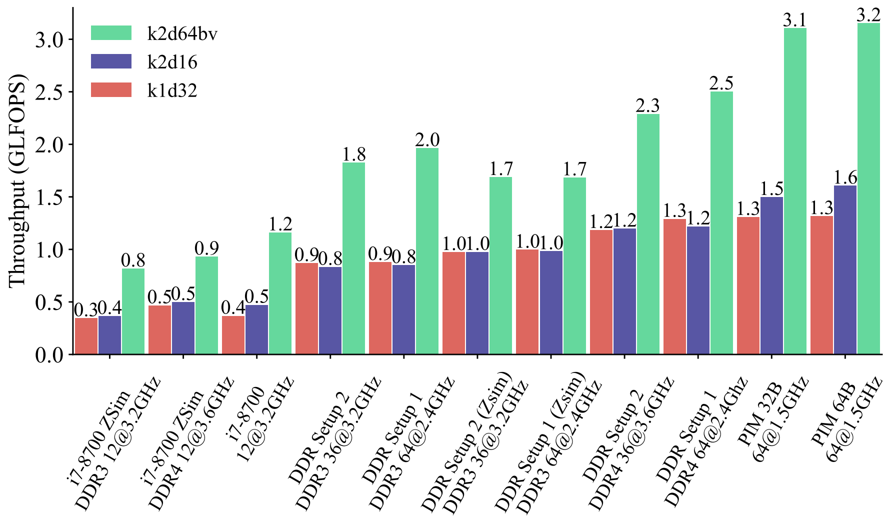


Figure B.11: Rendimiento de FM-index para varias arquitecturas

Finalmente, la Figura B.13 muestra tanto el ancho de banda aleatorio como el número de operaciones LF realizadas por cada Julio de energía utilizado. Se puede observar que la eficiencia energética es muy superior para las configuraciones que utilizan arquitecturas PIM con núcleos de alta eficiencia energética y bajo consumo. En concreto, las arquitecturas PIM son capaces de realizar hasta 8 veces más operaciones LF por julio y hasta 10 veces más accesos aleatorios que el sistema con un i7-7800. Además, estas arquitecturas consiguen una mejora aún mayor al compararlos con los sistemas con 32 y 64 núcleos, alcanzando una eficiencia energética entre 21 y 40 veces superior.

B.4 Bowtie2 y Procesado en Memoria

En este capítulo presentamos una exploración arquitectural para aplicaciones con accesos de memoria aleatorios, utilizando específicamente Bowtie2, una aplicación popular de alineamiento de secuencias en genómica, como caso de estudio para el análisis de la eficiencia energética de distintos sistemas.

Como estas aplicaciones están principalmente limitadas por memoria, proponemos usar núcleos energéticamente eficientes ARMv8 de 64 bits. En el caso

UNIVERSIDAD DE MÁLAGA

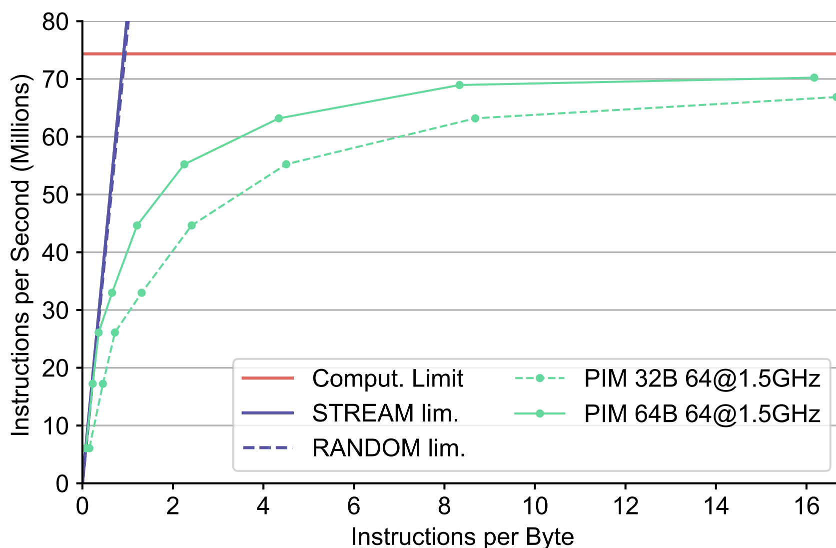


Figure B.12: Modelo Roofline para arquitectura PIM

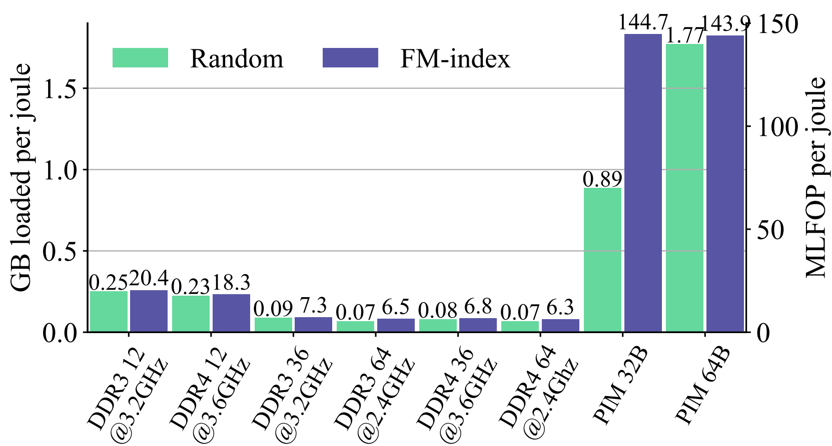


Figure B.13: Ancho de banda con accesos aleatorios y operaciones LF por julio de energía

UNIVERSIDAD DE MÁLAGA

del sistema de memoria, proponemos el uso de memoria 3D (HBM2), en lugar de la tradicional memoria DDR, como DDR4. También comparamos el rendimiento de sistemas basados en ARM con un procesador Intel Xeon Phi 7210 KNL, que incluye una memoria 3D MCDRAM en el mismo chip. La exploración arquitectural se lleva a cabo utilizando el simulador gem5-X [90], una versión extendida y validada del simulador ampliamente usado gem5 [15].

B.4.1 Bowtie2: Aplicación Completa de Alineamiento de Secuencias

Bowtie2 [64] es una aplicación de alineamiento de secuencias de código abierto, rápida y eficiente, utilizada para alinear cadenas de ADN con grandes genomas. También soporta alineamientos con huecos.

Bowtie2 está basado en la transformada de Burrows-Wheeler y el algoritmo FM-index para eficientemente encontrar alineamientos no exactos que satisfagan una política de alineamiento concreta. El algoritmo de Bowtie2 se divide en 4 pasos (ver Figura B.14).

Los índices de esta aplicación están optimizados con el objetivo de utilizar la mínima memoria posible. De esta forma, los índices de Bowtie2 del genoma humano usa en torno a 3.25GB en disco, y en memoria principal usa en torno a 1.3GB.

Al compararlo con otras herramientas de alineamiento, Bowtie2 es en torno a 2.5-3 veces más rápido que BWA (Burrows Wheeler Aligner) cuando ambas aplicaciones buscan alineamientos permitiendo huecos.

B.4.2 Entorno de Simulación Arquitectural

Nuestro entorno de simulación, basado en gem5-X nos permite realizar una exploración arquitectural relativamente rápida para arquitecturas optimizadas en energía y rendimiento, a nivel de sistema para cualquier aplicación. Algunos de los parámetros o variables con los que hemos trabajado para el estudio de rendimiento han sido:

Sistema de memoria: Proponemos el uso de memoria de alto rendimiento como HBM2. Hemos comparado este tipo de memoria novedosa con memorias más tradicionales como DDR.

UNIVERSIDAD DE MÁLAGA

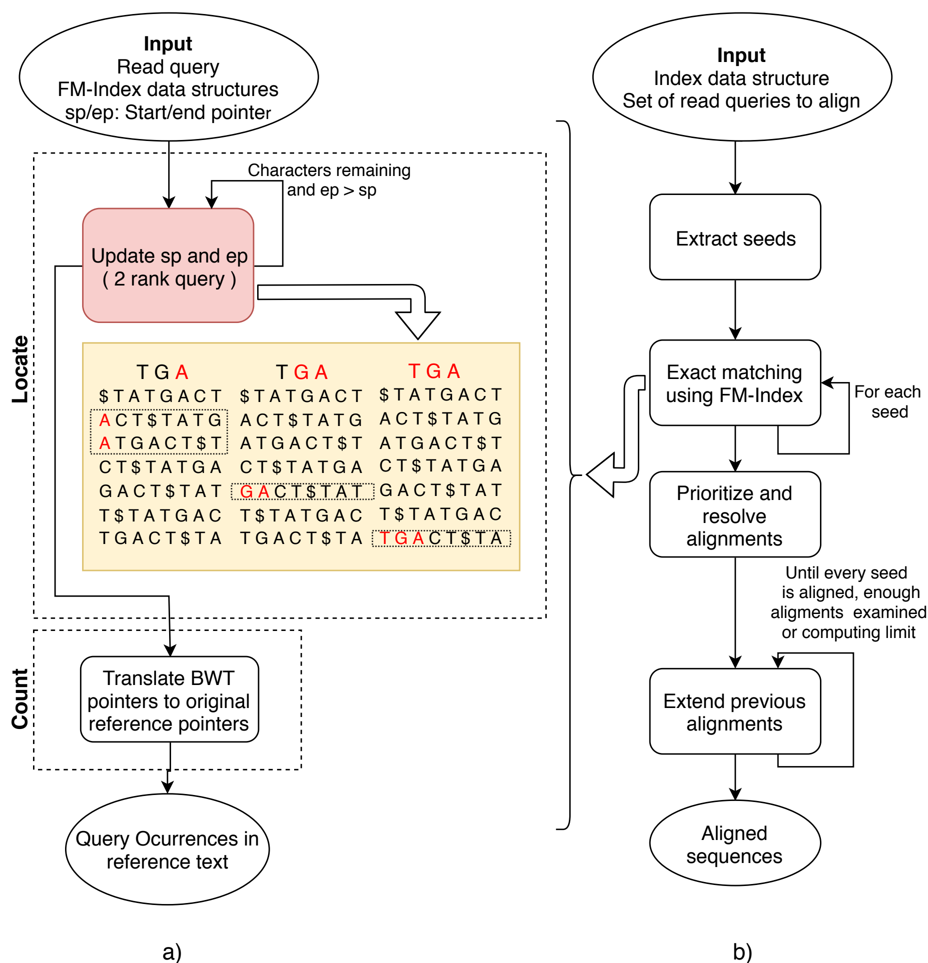


Figure B.14: Fases de los algoritmos de (a) FM-index y (b) Bowtie2

Tipo de núcleos: Comparamos la eficiencia tanto en rendimiento como energéticamente de núcleos fuera de orden con núcleos en orden, normalmente de menos rendimiento pero más eficientes en términos de energía.

Número de núcleos: Hemos explorado cómo escala el rendimiento y el uso de energía con distinto número de núcleos, desde 8 hasta 28, comparando pocos núcleos fuera de orden con muchos en orden.

Frecuencia de procesadores: Analizamos distintas frecuencias para los proce-

UNIVERSIDAD DE MÁLAGA

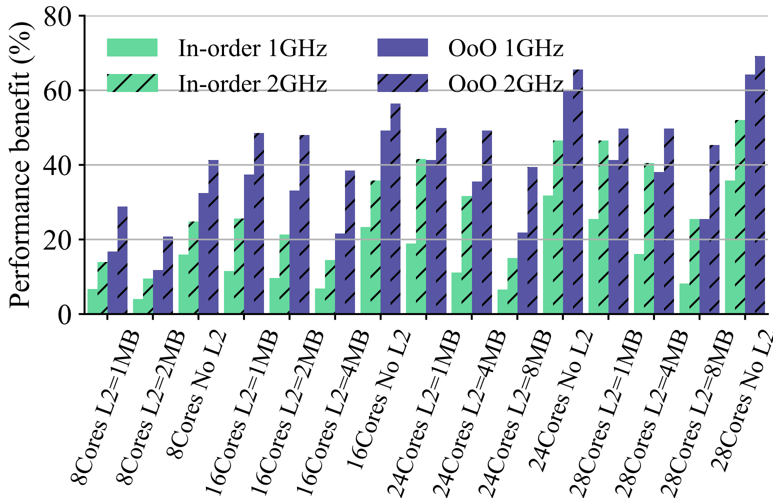


Figure B.15: Mejora de rendimiento de HBM2 vs DDR4

sadores, entre 1GHz y 2GHz.

Cache de último nivel: Por último, hemos probado diferentes configuraciones de la cache de último nivel (L2) con distintos tamaños entre 1MB y 8MB.

B.4.3 Resultados y Discusión

Hemos llevado a cabo numerosos experimentos ejecutando Bowtie2 en gem5-X. Las Figuras B.15 y B.16 muestran comparativas de rendimiento entre arquitecturas HBM2 y DDR4 con diferentes configuraciones cache. En estas figuras se puede ver que HBM2 proporciona una mejora en términos de eficiencia energética y rendimiento de aproximadamente un 70% cuando comparamos estas dos arquitecturas sin usar L2. En un sistema con cache L2, esta mejora se reduce hasta un 50% en rendimiento y un 56% en eficiencia energética.

Por otro lado, las Figuras B.17 y B.18 muestra el escalado de rendimiento y energía a 1GHz con diferentes configuraciones y diferente número de núcleos. Podemos observar como muchos núcleos en orden sencillos son capaces de superar a una cantidad inferior de cores fuera de orden, en términos tanto de rendimiento como de energía, usando un área igual o menor.

UNIVERSIDAD DE MÁLAGA

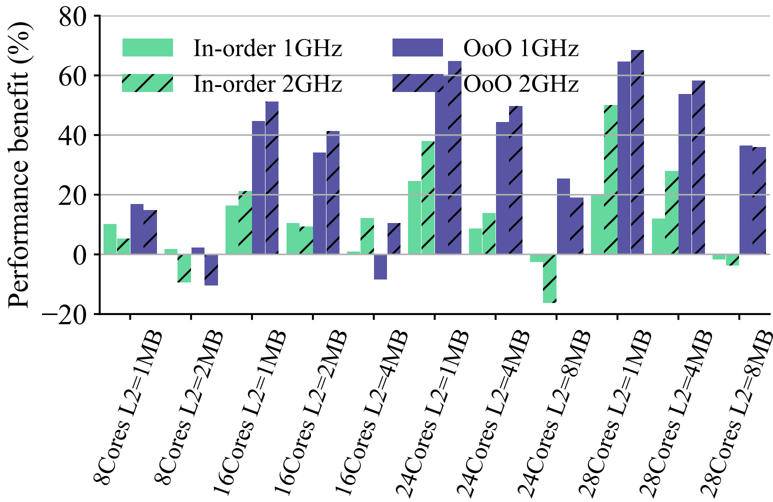


Figure B.16: Mejora de rendimiento de HBM2 sin L2 vs DDR4 con L2

B.5 Conclusiones

En esta tesis hemos tratado la optimización de aplicaciones intensivas en memoria con distintas arquitecturas, tanto reales como simuladas.

El capítulo 3 analiza varias versiones de FM-index y propone una variación sobre ellas, con una mejora importante de rendimiento. Nuestra versión supera a la mejor de las versiones previas adaptadas a estos sistemas, en un 130% y 90% para KNL y SKL, respectivamente.

En el capítulo 4, analizamos el rendimiento de aplicaciones con accesos aleatorios en arquitecturas convencionales y arquitecturas PIM, consiguiendo un rendimiento de entre 2.7 y 3.7 veces comparando una configuración PIM con la configuración de 12 cores y entre 1.26 y 1.87 veces comparado con las configuraciones de 36 y 64 cores con memorias DDR3 y DDR4.

En último lugar, en el capítulo 5 usamos gem5-X para comparar el rendimiento de Bowtie2 en distintas arquitecturas. Usando memorias HBM2 rápidas con núcleos de cómputo muy eficientes conseguimos una mejorada de hasta un 68% en rendimiento y hasta un 71% en energía cuando se compara con un sistema tradicional con DDR4. También se demuestra que se puede conseguir hasta un

UNIVERSIDAD DE MÁLAGA

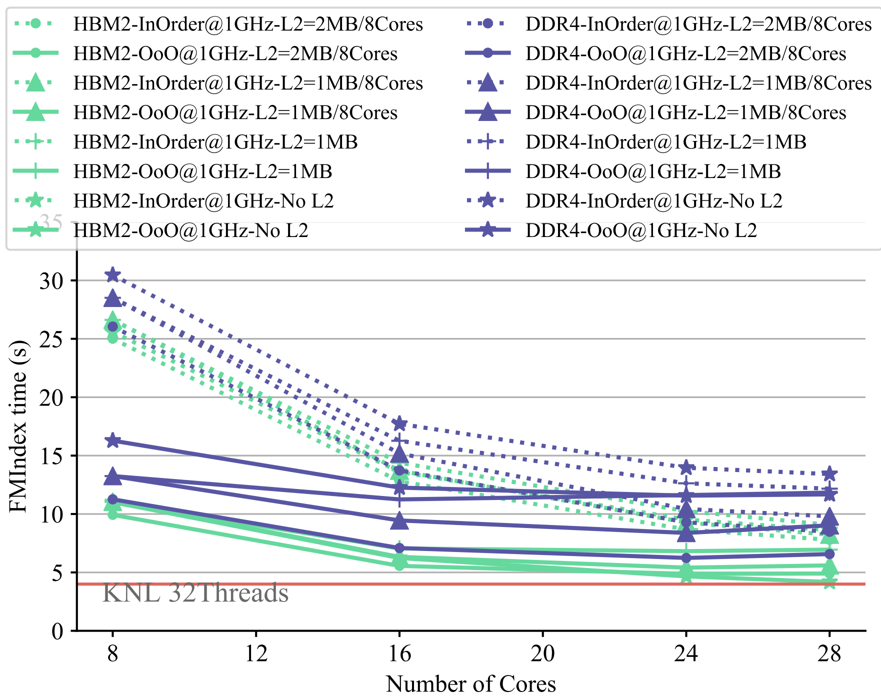


Figure B.17: Escalado de rendimiento a 1GHz

47% de ahorro de energía al utilizar muchos núcleos pequeños en orden, en lugar de menos núcleos fuera de orden.

UNIVERSIDAD DE MÁLAGA

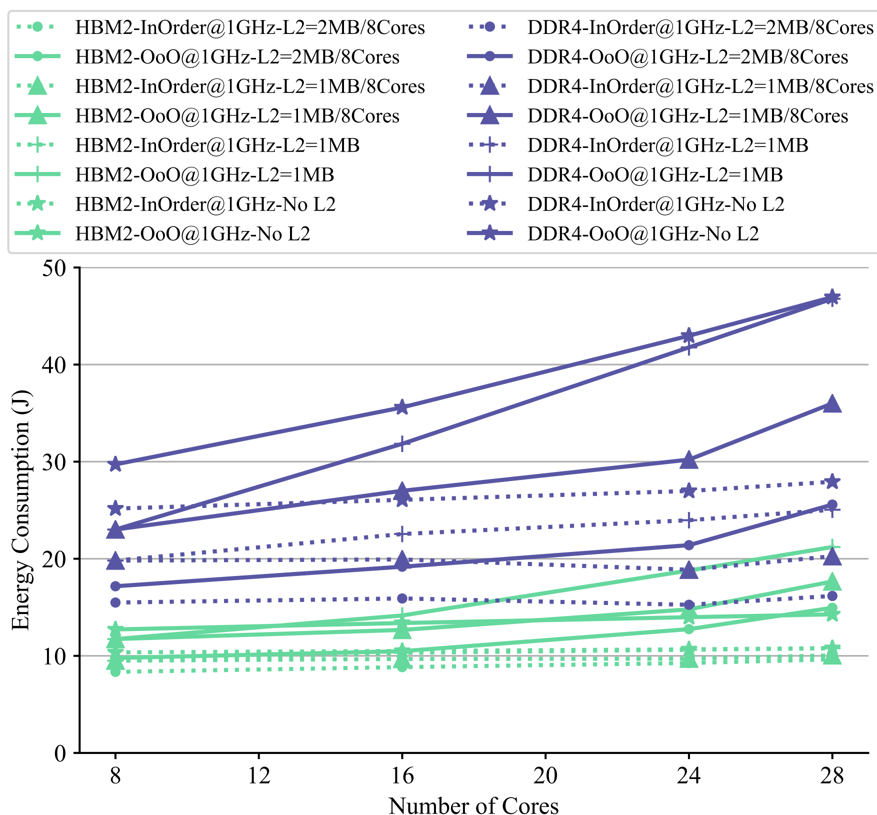


Figure B.18: Escalado de energía a 1GHz

Los resultados de esta tesis han derivado en las siguientes publicaciones:

Exact Alignment with FM-index on the Intel Xeon Phi Knights Landing Processor
Jose M. Herruzo, Sonia Gonzalez-Navarro, Pablo Ibañez, Victor Viñals, Jesus Alastruey and Oscar Plata
Workshop on Accelerator Architecture in Computational Biology and Bioinformatics (AACBB'18) (co-located with HPCA 2018), Vienna (Austria), February 2018.

UNIVERSIDAD DE MÁLAGA

Optimizing Large Data Structures with Unpredictable Access Patterns in the Intel KNL Processor
Jose M. Herruzo, Sonia Gonzalez-Navarro and Oscar Plata
20th Workshop on Compilers for Parallel Computing (CPC'18), Dublin (Ireland), April 2018.

Accelerating Sequence Alignments Based on FM-index Using the Intel KNL Processor.
Jose M. Herruzo, Sonia González, Pablo Ibáñez, Víctor Viñals, Jesús Alastruey-Benedé, and Óscar Plata
IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), December 2018.

Boosting Backward Search Throughput for FM-index Using a Compressed Encoding.
Jose M. Herruzo, Sonia González, Pablo Ibáñez, Víctor Viñals, Jesús Alastruey-Benedé, and Óscar Plata
Proceedings of the 2019 Data Compression Conference (DCC 2019), 26–29 March 2019, Snowbird, Utah, USA, pp. 577.

Aceleración de una Aplicación con Acceso Intensivo e Impredecible a los Datos en el Procesador Intel Xeon Phi KNL
Jose M. Herruzo, Sonia Gonzalez-Navarro, Pablo Ibañez, Victor Viñals, Jesus Alastruey and Oscar Plata
XXIX Jornadas de Paralelismo (part of Jornadas Sarteco), Teruel (Spain), September 2018.

Los resultados de las dos últimas secciones están actualmente en evaluación en una revista internacional y un congreso internacional, respecívamente.

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD
DE MÁLAGA

Bibliography

[1] Data centres and data transmission networks tracking clean energy progress. https://www.iea.org/tcep/buildings/datacentres/. Retrieved: September 2019.

[2] Cortex-a35 – arm developer. https://developer.arm.com/ip-products/processors/cortex-a/cortex-a35. Retrieved: September 2019.

[3] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo. BigBWA: Approaching the Burrows-Wheeler aligner to big data technologies. *Bioinformatics*, 31(24):4003–4005, 2015.

[4] Advanced Micro Devices, Inc. High Bandwidth Memory (HBM). http://www.amd.com/en/technologies/hbm. Retrieved: September 2019.

[5] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. A scalable processing-in-memory accelerator for parallel graph processing. In *Int'l. Symp. on Computer Architecture (ISCA'15)*, pages 105–117, 2015.

[6] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture. In *Int'l. Symp. on Computer Architecture (ISCA'15)*, pages 336–348, 2015.

[7] Ayaz Akram and Lina Sawalha. A comparison of x86 computer architecture simulators. *Faculty Research and Creative Activities Award (FRACAA) Recipients*, 2016.

[8] António Anjos, Ricardo Leite, M. Leonor Cancela, and Hamid Shahbazkia. Maq – a bioinformatics tool for automatic macroarray analysis. *International Journal of Computer Applications*, 4, 07 2010.

UNIVERSIDAD DE MÁLAGA

[9] ARM. ARM versatile express juno r2 development platform, 2015.

[10] Ryo Asai. MCDRAM as High-Bandwidth Memory (HBM) in Knights Landing processors: Developer's guide. `https://colfaxresearch.com/knl-mcdram`, 2016.

[11] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems. In *Int'l. Symp. on Microarchitecture (MICRO'49)*, pages 50:1–50:13, 2016.

[12] M. Usman Ashraf, Fathy Eassa, A.A. Albeshri, and Abdullah Algarni. Toward exascale computing systems: An energy efficient massive parallel computational model. *IJACSA*, pages 118–126, January 2018.

[13] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. Near-data processing: Insights from a micro-46 workshop. *IEEE Micro*, 34(4):36–42, 2014.

[14] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[15] Nathan Binkert et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, pages 1–7, August 2011.

[16] Bryan Black, Murali Annavaram, Ned Brekelbaum, John DeVale, Lei Jiang, Gabriel H Loh, Don McCaule, Pat Morrow, Donald W Nelson, Daniel Pantuso, et al. Die stacking (3d) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–479. IEEE Computer Society, 2006.

[17] Jessica S. Black, Manuel Salto-Tellez, Ken I. Mills, and Mark A. Catherwood. The impact of next generation sequencing technologies on haematological research – a review. *Pathogenesis*, pages 9 – 16, 2015.

[18] J Blom, T Jakobi, D Doppmeier, S Jaenicke, J Kalinowski, J Stoye, and Goesmann A. Exact and complete short-read alignment to microbial genomes using Graphics Processing Unit programming. *Bioinformatics*, 27(10):1351–1358, 2011.

UNIVERSIDAD DE MÁLAGA

[19] Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu. Google workloads for consumer devices: Mitigating data movement bottlenecks. *SIGPLAN Not.*, 53(2):316–331, 2018.

[20] Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh, Nastaran Hajinazar, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu. Conda: Efficient cache coherence support for near-data accelerators. In *Int'l. Symp.s on Computer Architecture (ISCA'19)*, pages 629–642, 2019.

[21] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[22] Alejandro Chacón, Santiago Marco-Sola, Antonio Espinosa, Paolo Ribeca, and Juan Carlos Moure. Boosting the FM-index on the GPU: Effective techniques to mitigate random memory access. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pages 1048–1059, 2015.

[23] Alejandro Chacón, Juan Carlos Moure, Antonio Espinosa, and Porfidio Hernandez. n-step FM-index for faster pattern matching. *Procedia Computer Science*, 18:70–79, 2013.

[24] Alejandro Chacón, Santiago Marco Sola, Antonio Espinosa, Paolo Ribeca, and Juan Carlos Moure. FM-index on GPU: A cooperative scheme to reduce memory footprint. In *IEEE Int. Symp. on Parallel and Distributed Processing with Applications (ISPA 2014)*, 2014.

[25] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.

[26] Su Chen and Hai Jiang. An exact matching approach for high throughput sequencing based on BWT and GPUs. *IEEE 14th Int. Conf. on Computational Science and Engineering*, 2011.

[27] Roxana Cojocneanu, Laura Pop, Ancuta Jurj, Lajos Raduly, Dan Dumitrascu, Nicolae Dragos, and Ioana Berindan Neagoe. Next generation sequencing applications for breast cancer research. *Clujul Medical*, 88:278, 07 2015.

UNIVERSIDAD DE MÁLAGA

[28] Douglas Doerfler, Jack Deslippe, Samuel Williams, Leonid Oliker, Brandon Cook, Thorsten Kurth, Mathieu Lobet, Tareq Malas, Jean-Luc Vay, and Henri Vincenti. Applying the roofline performance model to the Intel Xeon Phi Knights Landing processor. In *Int. Conf. on High Performance Computing*, 2016.

[29] Mario Paulo Drumond Lages De Oliveira, Alexandros Daglis, Nooshin Mirzadeh, Dmitrii Ustiugov, Javier Picorel Obando, Babak Falsafi, Boris Grot, and Dionisios Pnevmatikatos. The mondrian data engine. *Int'l. Symp. on Computer Architecture (ISCA'17)*, 2017.

[30] Duncan Elliott, Michael Stumm, W. Martin Snelgrove, Christian Cojocaru, and Robert McKenzie. Computational RAM: Implementing processors in memory. *IEEE Des. Test*, 16(1):32–41, 1999.

[31] Amin Farmahini Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. NDA: Near-DRAM acceleration architecture leveraging commodity DRAM devices and standard memory modules. *Int'l. Symp. on High Performance Computer Architecture (HPCA'15)*, pages 283–295, 2015.

[32] Edward B. Fernandez, Jason Villarreal, and Stefano Lonardi. FHAST: FPGA-based acceleration of Bowtie in hardware. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 12(5):973–981, 2015.

[33] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st. Ann. Symp. on Foundations of Computer Science*, pages 390–398, 2000.

[34] Luca Foschini, Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. When indexing equals compression: Experiments with compressing suffix arrays and applications. *ACM Transactions on Algorithms*, 2(4):611–639, 2006.

[35] A. Frumusanu. "arm announces new cortex-a35 cpu - ultra-high efficiency for wearables & more". `https://www.anandtech.com/show/9769/arm-announces-cortex-a35`, 2015.

[36] A. Frumusanu and R. Smith. "Cortex A53 - performance and power". `https://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/4`, 2015.

[37] A. Frumusanu and R. Smith. "Cortex A57 - performance and power". `https://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/6`, 2015.

UNIVERSIDAD DE MÁLAGA

[38] Travis Gagie, Gonzalo Navarro, and Simon J. Puglisi. New algorithms on wavelet trees and applications to information retrieval. *Theoretical Computer Science*, 426–427:25–41, 2012.

[39] Mingyu Gao, Grant Ayers, and Christos Kozyrakis. Practical near-data processing for in-memory analytics frameworks. In *Int'l. Conf. on Parallel Architectures and Compilation (PACT'15)*, pages 113–124, 2015.

[40] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. TETRIS: scalable and efficient neural network acceleration with 3d memory. In *Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'17)*, pages 751–764, 2017.

[41] Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, and Onur Mutlu. Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions. *CoRR*, abs/1802.00320, 2018.

[42] Simon Gog, Juha Kärkkäinen, Dominik Kempa, Matthias Petry, and Simon J. Puglisi. Faster, minuter. In *Data Compression Conf. (DCC 2016)*, 2016.

[43] Simon Gog and Matthias Petri. Optimized succinct data structures for massive data. *Software – Practice & Experience*, 44(11), 2014.

[44] Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: A tool for text indexing. In *17th Ann. ACM-SIAM Symp. on Discrete Algorithm (SODA 2006)*, pages 368–373, 2006.

[45] Jorge Gonzalez-Dominguez, Yongchao Liu, and Bertil Schmidt. Parallel and scalable short-read alignment on multi-core clusters using UPC++. *PLoS One*, 11(1), 2016.

[46] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *14th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 2003)*, pages 841–850, 2003.

[47] Seth H Pugsley, Jeffrey Jestes, Huihui Zhang, Rajeev Balasubramonian, Vijayalakshmi Srinivasan, Alper Buyuktosunoglu, Al Davis, and Feifei Li. NDC: Analyzing the impact of 3d-stacked memory+logic devices on mapreduce workloads. In *Int'l. Symp. on Performance Analysis of Systems and Software (ISPASS'14)*, pages 190–200, 2014.

UNIVERSIDAD DE MÁLAGA

[48] M. Hall, P. Kogge, J. Koller, P. Diniz, J. Chame, J. Draper, J. LaCoss, J. Granacki, J. Brockman, A. Srivastava, W. Athas, V. Freeh, Jaewook Shin, and Joonseok Park. Mapping irregular applications to DIVA, a PIM-based data-intensive architecture. In *Conf. on Supercomputing (SC'99)*, pages 57–57, 1999.

[49] M Holtgrewe. Mason - a read simulator for second generation sequencing data. Technical Report 962, Freie Universitaet Berlin, 2010.

[50] K. Hsieh, E. Ebrahim, G. Kim, N. Chatterjee, M. O'Connor, N. Vijaykumar, O. Mutlu, and S. W. Keckler. Transparent offloading and mapping (TOM): Enabling programmer-transparent near-data processing in GPU systems. In *Int'l. Symp. on Computer Architecture (ISCA'16)*, pages 204–216, 2016.

[51] K. Hsieh et al. Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation. In *ICCD*, pages 25–32, 2016.

[52] Intel® core™ i7-8700 processor (12m cache, up to 4.60 ghz) product specifications. `https://ark.intel.com/content/www/us/en/ark/products/126686/intel-core-i7-8700-processor-12m-cache-up-to-4-60-ghz.html`. Retrieved: September 2019.

[53] Intel architecture code analyzer, intel software. `https://software.intel.com/en-us/articles/intel-architecture-code-analyzer`. Retrieved: September 2019.

[54] Guy Jacobson. Space-efficient static trees and graphs. In *30th Ann. Symp. on Foundations of Computer Science*, 1989.

[55] Joe Jeddeloh and Brent Keeth. Hybrid memory cube new DRAM architecture increases density and performance. In *Symp. on VLSI technology (VLSIT'12)*, pages 87–88, 2012.

[56] Joe Jeddeloh and Brent Keeth. Hybrid Memory Cube new DRAM architecture increases density and performance. In *Symp. on VSLI Technology*, pages 87–88, 2012.

[57] P. Ibáñez V. Viñals J. Alastruey J.M. Herruzo, S. Navarro-González and O. Plata. Accelerating sequence alignments based on fm-index using the intel knl processor. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB 2019)*, 2019.

UNIVERSIDAD DE MÁLAGA

[58] Daehwan Kim, Ben Langmead, and Steven L Salzberg. Hisat: a fast spliced aligner with low memory requirements. *Nature Methods*, 12, 2015.

[59] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory. *Comput. Archit. News*, 44(3):380–392, 2016.

[60] Jeremie S. Kim, Damla Senol, Hongyi Xin, Donghyuk Lee, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu. Genome read in-memory (GRIM) filter: Fast location filtering in DNA read mapping using emerging memory technologies, 2017. Pacific Symp. on Biocomputing (PSB'17).

[61] Yoongu Kim, Weikun Yang, and Onur Mutlu. Ramulator: A fast and extensible dram simulator. *IEEE Computer architecture letters*, 15(1):45–49, 2015.

[62] Petr Klus, Simon Lam, Dag Lyberg, Ming Sin Cheung, Graham Pullan, Ian McFarlane, Giles S H Yeo, and Brian Y H Lam. BarraCUDA - a fast short read sequence aligner using graphics processing units. *BMC Research Notes*, 5(27), 2012.

[63] T W Lam, W K Sung, S L Tam, C K Wong, and S M Yiu. Compressed indexing and local alignment of DNA. *Bioinformatics*, 6(24):791–797, 2008.

[64] Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, 2012.

[65] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25.1–R25.10, 2009.

[66] Ben Langmead, Christopher Wilks, Valentin Antonescu, and Rone Charles. Scaling read aligners to hundreds of threads on general-purpose processors. *Bioinformatics*, pages 421–432, July 2018.

[67] M. A. Laurenzano et al. Characterization and bottleneck analysis of a 64-bit ARMv8 platform. In *ISPASS*, pages 36–45, 2016.

[68] S. Lee et al. Leveraging power-performance relationship of energy-efficient modern DRAM devices. *IEEE Access*, pages 31387–31398, 2018.

[69] Heng Li and R. Durbi. Fast and accurate long read alignment with Burrows-Wheeler transform. *Bioinformatics*, 5(26):589–595, 2010.

UNIVERSIDAD DE MÁLAGA

[70] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[71] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–483, May 2010.

[72] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karten Kristiansen, and Jun Wang. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

[73] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. ACM, 2009.

[74] Yongchao Liu and Bertil Schmidt. CUSHAW2-GPU: Empowering faster gapped short-read alignment using GPU computing. *IEEE Design and Test*, 31(1):31–39, 2014.

[75] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200, June 2005.

[76] R. Luo, T. Wong, J. Zhu, C.-M. Liu, X. Zhu, E. Wu, L.-K. Lee, H. Lin, Cheung D. W. Zhu, W., H.-F. Ting, S.-M. Yiu, S. Peng, C. Yu, Y. Li, R. Li, and T.-W. La. SOAP3-dp: Fast, accurate and sensitive GPU-based short read aligner. *PLoS One*, 8, 2013.

[77] Veli Mäkinen and Gonzalo Navarro. Rank and select revisited and extended. *Theoretical Computer Science*, 387(3), 2007.

[78] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *1st Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA 1990)*, pages 319–327, 1990.

[79] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[80] John D. McCalpin. Memory bandwidth and machine balance in high performance computers. *IEEE Technical Committee on Computer Architecture Newsletter*, pages 19–25, December 1995.

UNIVERSIDAD DE MÁLAGA

[81] Micron Technology, Inc. Hybrid Memory Cube (HMC). https://www.micron.com/products/hybrid-memory-cube. Retrieved: September 2019.

[82] ”tn-40-07: Calculating memory power for ddr4 sdram”. https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf. Retrieved on November 2019.

[83] Makoto Motoyoshi. Through-silicon via (tsv). *Proceedings of the IEEE*, 97(1):43–48, 2009.

[84] Lifeng Nai, Ramyad Hadidi, Jaewoong Sim, Hyojong Kim, Pranith Kumar, and Hyesoon Kim. GraphPIM: Enabling instruction-level PIM offloading in graph computing frameworks. In *Int'l. Symp. on High Performance Computer Architecture (HPCA'17)*, pages 457–468, 2017.

[85] NVBIO: A library of reusable components designed by NVIDIA corporation to accelerate bioinformatics applications using CUDA. http://nvlabs.github.io/nvbio. Retrieved: September 2019.

[86] Mike O'Connor et al. Fine-grained dram: Energy-efficient DRAM for extreme bandwidth systems. In *MICRO*, pages 41–54, 2017.

[87] Z. Ou et al. Energy- and cost-efficiency analysis of ARM-based clusters. In *CCGRID*, pages 115–123, 2012.

[88] A. Pahlevan et al. Energy proportionality in near-threshold computing servers and cloud data centers: Consolidating or not? In *DATE*, pages 147–152, 2018.

[89] I. B. Peng et al. Exploring the performance benefit of hybrid memory system on HPC environments. In *IPDPSW*, pages 683–692, 2017.

[90] Y. M. Qureshi, W. A. Simon, M. Zapater, D. Atienza, and K. Olcoz. Gem5-X: A gem5-based system level simulation framework to optimize many-core platforms. In *2019 SpringSim*, pages 1–12, April 2019.

[91] M. Radulovic et al. Mainstream vs. emerging HPC: Metrics, trade-offs and lessons learned. In *SBAC-PAD*, pages 250–257, 2018.

[92] N. Rajovic et al. Supercomputing with commodity cpus: Are mobile socs ready for HPC? In *SC*, pages 1–12, 2013.

UNIVERSIDAD DE MÁLAGA

[93] Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4), 2007.

[94] Jennifer Ronholm. Editorial: Game changer - next generation sequencing and its impact on food microbiology. *Frontiers in Microbiology*, page 363, 2018.

[95] Roofline performance model. `http://crd.lbl.gov/departments/computer-science/PAR/research/roofline`. Retrieved: September 2019.

[96] Ruby - gem5. `http://gem5.org/Ruby`. Retrieved: October 2019.

[97] Daniel Sanchez and Christos Kozyrakis. Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. In *ACM SIGARCH Computer architecture news*, volume 41, pages 475–486. ACM, 2013.

[98] Bertil Schmidt and Andreas Hildebrandt. Next-generation sequencing: big data meets high performance computing. *Drug Discovery Today*, pages 712 – 717, 2017.

[99] Succinct data structure library 2.0. `https://github.com/simongog/sdsl-lite`. Retrieved: September 2019.

[100] Andreas Selinger, Karl Rupp, and Siegfried Selberherr. Evaluation of mobile ARM-based socs for high performance computing. In *HPC*, pages 1–7, 2016.

[101] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *HPCCS – VECPAR 2010*, pages 1–25, 2011.

[102] Julian Shun and Guy E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. *SIGPLAN Not.*, pages 135–146, February 2013.

[103] Gagandeep Singh, Juan Gómez-Luna, Giovanni Mariani, Geraldo F Oliveira, Stefano Corda, Sander Stuijk, Onur Mutlu, and Henk Corporaal. Napel: Near-memory computing application performance prediction via ensemble learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 27. ACM, 2019.

[104] A. Sodani et al. Knights landing: Second-generation intel xeon phi product. *IEEE Micro*, pages 34–46, Mar 2016.

UNIVERSIDAD DE MÁLAGA

[105] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. Knights Landing: Second-generation Intel Xeon Phi product. *IEEE Micro*, 36(2):34–46, March 2016.

[106] K. Sohn et al. A 1.2 v 20 nm 307 GB/s HBM DRAM with at-speed wafer-level io test scheme and adaptive refresh considering temperature distribution. *JSSC*, pages 250–260, Jan 2017.

[107] Harold S Stone. A logic-in-memory computer. *IEEE Transactions on Computers*, 100(1):73–78, 1970.

[108] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Ann. Int. Symp. on Computer Architecture (ISCA 1995)*, June 1995.

[109] Andrey Vladimirov and Ryo Asai. Clustering modes in Knights Landing processors: Developer's guide. `https://colfaxresearch.com/knl-numa`, 2016.

[110] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, April 2009.

[111] Richard Wilton, Tamas Budavari, Ben Langmead, Sarah J. Wheelan, Steven L. Salzberg, and Alexander S. Szalay. Arioc: high-throughput read alignment with GPU-accelerated exploration of the seed-and-extend search space. *PeerJ*, 3:e808, 2015.

[112] Richard Wilton et al. Arioc: high-throughput read alignment with GPU-accelerated exploration of the seed-and-extend search space. *PeerJ*, page e808, March 2015.

[113] Wm A Wulf and Sally A McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[114] Intel® xeon® gold 6154 processor (24.75m cache, 3.00 ghz) product specifications. `https://ark.intel.com/content/www/us/en/ark/products/120495/intel-xeon-gold-6154-processor-24-75m-cache-3-00-ghz.html`. Retrieved: September 2019.

[115] Intel® xeon phi™ processor 7210 (16gb, 1.30 ghz, 64 core) product specifications. `https://ark.intel.com/content/www/us/en/ark/products/`

UNIVERSIDAD DE MÁLAGA

94033/intel-xeon-phi-processor-7210-16gb-1-30-ghz-64-core.html. Retrieved: September 2019.

[116] Bonan Zhang. Guide to automatic vectorization with Intel AVX-512 instructions in Knights Landing processors. https://colfaxresearch.com/knl-avx512, 2016.

[117] Dongping Zhang, Nuwan Jayasena, Alexander Lyashevsky, Joseph L Greathouse, Lifan Xu, and Michael Ignatowski. TOP-PIM: throughput-oriented programmable processing in memory. In *Int'l. Symp. on High-performance Parallel and Distributed Computing (HPDC'14)*, pages 85–98, 2014.

[118] Jun Zhang, Rod Chiodini, Ahmed Badr, and Genfa Zhang. The impact of next-generation sequencing on genomics. *Journal of Genetics and Genomics*, pages 95 – 109, 2011.

[119] Mingxing Zhang, Youwei Zhuo, Chao Wang, Mingyu Gao, Yongwei Wu, Kang Chen, Christos Kozyrakis, and Xuehai Qian. GraphP: Reducing communication for PIM-based graph processing with efficient data partition. In *Int'l. Symp. on High Performance Computer Architecture (HPCA'18)*, pages 544–557, 2018.

[120] s5z/zsim: A fast and scalable x86-64 multicore simulator. https://github.com/s5z/zsim. Retrieved: September 2019.

UNIVERSIDAD DE MÁLAGA