DOI:10.15774/PPKE.ITK.2016.007

Memory Access Optimization for Computations on Unstructured Meshes

Thesis submitted for the degree of Doctor of Philosophy

Antal Hiba M.Sc.

Supervisors:

Dr. Péter Szolgay, Dr. Miklós Ruszinkó

Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Roska Tamás Doctoral School of Sciences and Technology

Budapest, 2015

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

Contents

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

Kivonat

A gráfok rendkívül hatékony eszköznek bizonyultak a világ matematikai modellezésében, kezdve a közúti hálózatoktól egészen egy emberi ér 3D térbeli leírásáig. A modellekből származó numerikus algoritmusok gyakran számításigényesek, ezért több processzáló csomópont között fel kell osztani a feladatot. A felosztás során egy gráf particionálási feladatot kell megoldani. A cél a lehető leggyorsabb párhuzamosított számítás az erőforrások maximális kihasználásával. Az egymagos lassú kommunikációs csatornákkal összekötött processzorok idejében egyértelműen a minimális kommunikációt eredményező azonos méretű részgráfokat tartalmazó felosztásokat tekinthettük optimálisnak. Napjaink processzor-architektúrái azonban akár sok ezer műveletvégző egységet is tartalmazhatnak, és kommunikációs csatornáik sávszélessége csak egy nagyságrenddel marad el a saját központi memóriaelérésük sávszélességétől. Kutatásom célja, hogy megvizsgáljam mely felosztások optimálisak az új kiloprocesszoros architektúrák számára, és természetesen az ilyen felosztások generálására módszereket is szeretnék megadni.

A dolgozatban bemutatom korunk legjobb processzorait, és ezek paraméterein keresztül a memória sávszélességkorlát általános problémáját. A sávszélességkorlát leküzdésének általam leghatékonyabbnak talált irányát az adatfolyam alapú architektúrákat (DM) részletesen is bemutatom. Ezen speciális processzorok FPGA-n valósíthatóak meg vagy VLSI célhardvareként. A DM-ek legfontosabb tulajdonsága, hogy a processzáló egységek egy maximális sávszélességgel beérkező adatfolyamra csatlakoznak, a részeredmények ugyanígy haladnak tovább a chip-en további proceszáló elemek felé majd a végeredmény vissza a központi memóriába. A DM-ek maximálisan kihasználják a memória és a processzor közötti sávszélességet, viszont az adatfolyamnak csak egy nagyon kis része érhető el egy adott időpillanatban (on-chip cache). Az egy

DOI:10.15774/PPKE.ITK.2016.007

művelethez tartozó bemeneti adatoknak közel kell lenniük egymáshoz a soros adatfolyamban, különben a művelet nem lesz elvégezhető (az egyik operandus még nem ér be, amikor a másikat már ki kell dobni a cache-ből). Az összefüggő adatok maximális távolságát a folyamban adatlokalitásnak nevezzük.

Mivel az adatlokalitás erős korlát, szükséges egy olyan módszer, amely egy adatlokalitás korlát teljesülését képes garantálni a létrejövő részgráfokban. Az adatlokalitás javítása a mátrix sávszélesség minimalizálás feladatára vezethető vissza. Az irodalomban talált heurisztikus módszerek vizsgálata után, a Gibbs-Pole-Stockmeyer (GPS) módszert alakítottam át olyan particionálási eljárássá (AM1), amely adott mátrix sávszélesség korlát alatt sorosítható részgráfokat hoz létre. Az AM1 lehetővé teszi a DM-ek alkalmazását nagyobb részgráfokra is, viszont a részgráfok számát nem lehet előírni, és a létrejövő részgráfok mérete szintén nem egységesíthető. Az AM1 önmagában csak az egy DM által kezelhető részgráf méretét növeli meg szignifikánsan egy virtuális particionálás segítségével.

Az adatlokalitás matematikai vizsgálata több egyszerű összefüggést is eredményezett az adatlokalitás szempontjából kedvező partíciókkal kapcsolatban. Ezek alapján bevezettem a Mélységi Szint Struktúrát (DLS), amely a gráf végpontjaiból (vagy határhalmazból) indított szélességi keresés hoz létre. Elsőként biszekcióval foglalkoztam, és a legmélyebb szinteket szétvágó szeparátorokat generáló módszert dolgoztam ki. A létrejövő részgráfok adatlokalitás szempontból jól viselkedtek, viszont a proceszorközi kommunikáció megnövekedése és az irreguláris rácsok változatossága miatt a módszer nem lett eredményes.

A strukturált 2D és 3D téglalap vizsgálata során sikerült egy közel optimális módszert kidolgozni, amelyben az adatlokalitás és a processzorközi kommunikáció együttesen kezelhetővé vált. Ez alapján megfogalmaztam a Sávszélesség-korlátos Particionálás (BLP) feladatát. Megvizsgáltam hogyan lehetne korunk egyik legjobb particionálási módszerét (METIS) a BLP feladat megoldására felhasználni. Több izgalmas összefüggés mellett egy hibrid módszert kaptam, amely több METIS próba particionálás eredményéből kiszámol egy kezdeti particionálást, melynek elemeit átadja az AM1-nek.

DOI:10.15774/PPKE.ITK.2016.007

A párhuzamosítást segítő új particionálási módszerek mellett a dolgozat második részében magának az optimalizációnak a gyorsítását tűzöm ki célul. Maga a gráf particionálási feladat is felfogható egy kombinatorikus optimalizációs (CO) feladatként, ahol minden csomóponthoz tartozik egy döntési változó, amely meghatározza hogy hányas sorszámú részgráfba kerül az adott csomópont. Az ilyen feladatok megoldását úgy kívánom gyorsítani, hogy egy részfeladatot megoldok, és miközben a részmegoldást már felhasználom, folytatom a megmaradó döntési változók optimalizációját. A módszertant általánosan sikerült kidolgozni, és a Soros Rendezési Problémán (SOP) a Lemez Ütemezési Problémán (DSP) és az Általános Hozzárendelési Problémán (GAP) be is mutatom az előnyeit és gyenge pontjait. Sajnos a BLP megoldásának gyorsítására nem ajánlható a módszer, viszont sok más esetben, ahol egy részmegoldás ismerete nagyobb előnyökkel jár (feladatok sorbarendezése - SOP) jelentős gyorsítás érhető el.

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

Abstracts

Graphs have been found an effective tool in modeling the real world from the road network to the 3D volumetric description of a human vein. Numerical algorithms that come from these models are often computationally expensive, thus these tasks have to be divided. This division is defined by the solution of the graph partitioning problem. The partitioning aims for the best possible parallelism with maximal resource utilization. In the age of slow interconnections and single-core processors, the optimal solution was clearly the one with minimal communication need and identically sized parts. However, the current processor architectures have thousands of processing elements, and the bandwidth of their interconnections is just one order of magnitude slower than the bandwidth of their main memory. The goal of my research is to investigate the properties of optimal partitions in the case of these novel processor architectures and naturally, I want to also give methods to generate these partitions. In this work, I show the best processor architectures available today and through them the general problem of memory bandwidth limitation. I present in details Dataflow Machines (DM) which provides the best solution to this problem in my view. These specific processors can be realized in FPGA or custom VLSI. The most important property of DMs is streaming, that is, the processing elements connected to a maximal bandwidth data flow and the intermediate results propagated the same way to other processing elements on-chip and finally the result is streamed back to the main memory. DMs utilize the whole memory bandwidth, but only a small portion of the data flow can be accessed on-chip at the same time. The operands of an operation have to be close to each other in the serialized data stream, otherwise, the operation can not be executed (one of the operands has not arrived when an other operand must be thrown out from the cache). The maximal distance of connected data elements is called data locality.

DOI:10.15774/PPKE.ITK.2016.007

Data locality has a strict bound thus a method is required which can guarantee the desired locality in the resulting subgraphs. Data locality minimization of a stream can be traced back to the matrix bandwidth minimization problem. After the investigation of known methods, I modified the Gibbs-Pole-Stockmeyer (GPS) method into a partitioner (AM1), which creates subgraphs with the desired data locality. AM1 makes it possible to use DMs for larger subgraphs, however, the number of subgraphs can not be defined and the size of the subgraphs can not be controlled as well. AM1 itself can increase only the size of possible inputs for DMs with a virtual partitioning.
Mathematical analysis of data locality results many simple rules about the partitions with good data locality. Based on these observations I defined the Depth Level Structure (DLS), which is created by a Breadth-First Search started from the endpoints of the graph (or boundary elements). First, I work on bisections and I created a method which cuts the deepest areas of the graph. The resulted subgraphs have good data locality, however, the interprocesor communication need increased significantly and the variability of irregular meshes makes this approach unsuccessful.
Investigations of 2D and 3D structured rectangular meshes results a specific method which nearly optimal for these graphs, furthermore data locality and interprocessor communication can be handled together. Based on this result I can formulate the problem of Bandwidth Limited Partitioning (BLP). I do examinations with one of the best-known partitioners (METIS), how it can handle the BLP. I found some interesting connections, and I created a hybrid method which creates multiple METIS trials, and based on them creates an initial partition which is further partitioned with AM1.
After the improvements of parallelization, in the second part of this work I deal with the acceleration of the optimization itself. The graph partitioning problem is a Combinatorial Optimization (CO) problem, where each node has a decision variable that describes the index of the subgraph which will contain that node. I want to accelerate this optimization by the usage of partial solutions, before the optimization creates a complete solution. A general solver is defined and its derivative methods are shown for the Sequential Ordering Prob-

DOI:10.15774/PPKE.ITK.2016.007

lem (SOP), Disc Scheduling Problem (SDP) and the Generalized Assignment Problem (GAP) to demonstrate the advantages and highlight the weak points of the approach. Unfortunately, this approach is not beneficial for the BLP, but in other cases in which a partial solution gives more advantage (ordering tasks - SOP) it provides significant speedup.

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

Abbreviations

APSG	Applicable Partial Solution Generation
ASIC	Application-Specific Integrated Circuit
BFS	Breadth-First Search
BLP	Bandwidth Limited Partitioning
BW_Bound	Data Locality bound - Graph Bandwidth bound
C_BW	Central Bandwidth
CM	Cuthill McKee method
CO	Combinatorial Optimization
COMM_Bound	Computation over Communication Ratio bound
CPU	Central Processor Unit
DDR	Double Data Rate memory
DFE	Data-Flow Engine
DLS	Depth Level Structure
DM	Dataflow Machine
DMA	Direct Memory Access
DP	Double Precision
DPSO	Discrete Particle Swarm Optimization

DOI:10.15774/PPKE.ITK.2016.007

DRAM	Dynamic Random-access Memory
DSP	Digital Signal Processor
DSP	Disc Scheduling Problem
EACS	Enchanted Ant-Colony System
FIFO	First In First Out container
FLOPS	Floating-point Operation Per Second
FPGA	Field Programmable Gate Array
G_BW	Graph Bandwidth
GAP	Generalized Assignment Problem
GDDR	Graphics Double Data Rate memory
GPS	Gibbs Pole Stockmeyer method
GPU	Graphics Processing Unit
GRASP	Greedy Randomized Adaptive Search Procedure
MAC	Multiply-Accumulate
PCI	Peripheral Component Interconnect
PDE	Partial Differencial Equation
PE	Processing Element
S_BW	Serial Bandwidth
SIMD	Single Instruction Multiple Data
SOP	Sequential Ordering Problem
SP	Single Precision
TSP	Travelling Salesman Problem
VSM	Variable Subset Merger

DOI:10.15774/PPKE.ITK.2016.007

1. Chapter

Introduction

1.1. Motivation and scope

Recent processor architectures consist many parallel cores. These chips with high computational capacity are common and also the building blocks of current scientific and industrial supercomputers. The theoretical computational power reached 1 Tera-FLOPS/chip, but the utilization of such processors can be just 10-15% in a real-life application. The common reason is the difference between the memory bandwidth and the computational capacity of the processor. The speed of computation is not limited by the processing elements (PE), rather the memory bandwidth, which becomes a kind of wall between the processor and the memory [R1]. This effect is getting stronger because the computational capacity increases faster than the memory bandwidth. Several projects aim for a solution to the memory wall [R2] from these approaches I prefer the integration of memory with the processor array with 3D chip technology [R3, R4]. The Memory interface is not only a speed limiting factor, but it also increases the power consumption thus the optimized usage of these interfaces becomes important for economical and green computing reasons too.

Memory access optimization means the local memory (cache) handling methods of the processor, which decreases the number of transfers from/to the slow off-chip memory, and also includes the methods, which support the utilization of available off-chip memory bandwidth. The number of data transfers can be decreased by the on-chip cache if a data element can be reused during the computation. The utilization of available off-chip memory bandwidth can be increased by optimized memory access patterns. For the current DRAM technology, the serial access pattern is the most appropriate thus it is beneficial to

DOI:10.15774/PPKE.ITK.2016.007

organize the input data to reach serial access patterns during the computation. The task becomes more complex when the input problem is distributed among multiple processor chips. In this case, the inter-processor communication comes into the scene, which has to be considered to reach the best possible performance.

Classical distribution approaches consider only the interprocessor communication and identical part sizes. The elementary operators inside a task define a graph, where the nodes represent elementary operators, and the edges describe data dependencies. At this point, we reach the graph partitioning problem, which aims for identically sized subgraphs with minimal edge cut (communication need). The optimization of memory access is done by a node ordering algorithm, which increases data locality after the partitioning phase. Data locality maximization leads to the Matrix Bandwidth Minimization problem in which a matrix is transformed into a narrow banded form. Unfortunately, Graph Partitioning and Matrix Bandwidth Minimization are NP-complete [R5, R6] thus an optimal solution can not be obtained in polynomial time assuming P!=NP. Thanks to many successful research attempts there are several good heuristics to handle these problems. Because data locality becomes more and more important, a question arises: Should we consider data locality at the partitioning phase?

The primary goal of this dissertation is to investigate the connections between the partitions and the reachable data locality, and based on this knowledge, construct methods which can consider data locality and inter-processor communication at the same time. This approach can ensure better processor utilization and evade the wasting of resources. Parallelization and better memory bandwidth utilization require the solution of complex optimization tasks. Surely we do not want to spend much time with these optimization tasks. Novel efficient heuristics are often called metaheuristics which mean these methods are not simple task-specific heuristics, there is something 'more'. The possible solutions of a CO problem define a solution space. Metaheuristics provide dimension reduction (multilevel) or scouting techniques in the solution space (gradient, variable neighborhood search, simulated annealing, genetic algorithm, etc.) which description is task-independent [R7, R8]. In the case of graph partitioning, the dimension reduction methods are the most effective [R9], while for the matrix bandwidth minimization task-specific heuristics are used [R10, R11].

The second part of this dissertation makes an attempt to find new possibilities in dimension

DOI:10.15774/PPKE.ITK.2016.007

reduction, not limited to the graph partitioning problem.

1.2. Thesis outline

The thesis is organized as follows. In chapter 2, I show the best processor architectures available in 2013 and 2015 and through them the general problem of memory bandwidth limitation. In the following chapter, I present in details Dataflow Machines (DM) which provides the best solution to this problem in my view. Chapter 4 presents the existing variants of static mapping and introduces the problem of inter-processor communication and data locality in mesh partitioning. Chapter 5 contains the main results of bandwidth-limited mesh partitioning, which supports memory access optimization in the partitioning phase. The 6th chapter introduces a metaheuristic framework for fast-response combinatorial optimization. The last chapter summarizes the results and concludes the dissertation.

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

2. Chapter

Bandwidth limitations in mesh computing

Nowadays many-core architectures GPUs and FPGAs have hundreds of Processing Elements (PEs), which leads to high theoretical computational capacity (TeraFLOPS/chip). However, the utilization of PEs is low in many applications, because these architectures are very sensitive to irregular memory access patterns.

First of all, there is not enough theoretical memory bandwidth to feed all PEs simultaneously from off-chip memory. For utilizing all processing elements, loaded data must be reused several times from on-chip cache. Furthermore, the theoretical memory bandwidth can be reached only by sequential bursts (multiple data transfers together), and required data have to fit the provided access granularity (64 - 256 bit). In the case of random 32-bit reads, the real memory bandwidth can be many times lower than the theoretical maximum. Irregular memory access leads to poor memory bandwidth utilization, and high cache-miss rate, which are the sources of low PE utilization. Preoptimization of input data increases the regularity of the access pattern, but these effects still curtail PE efficiency.

This chapter gives some insight into the sources of memory bandwidth limitations and general solutions. Then, the case of mesh computing is discussed in more details. Dataflow Machines (DM) are introduced as a possible solution to the memory bandwidth limitation problem.

DOI:10.15774/PPKE.ITK.2016.007

2.1. Processors and Memory Interfaces

The theoretical computational power of processor architectures and the theoretical memory bandwidth of memory interfaces are increasing. However, the trends of their growth are different, and the gap between memory bandwidth and computational power become a performance bottleneck.

2.1.1. Comparison of different processor architectures from 2013

The processing capabilities and corresponding memory interfaces of processor chips are shown in Table 2.1. CPUs, GPUs and FPGAs have different purpose thus nobody can say that one is better than an other. The goal of this comparison is to show that all of these architectures suffer from the memory wall.

Intel core i7-4770K is a desktop CPU with 4 Haswell cores, where each core can perform 2x4 double precision (DP) floating-point multiply-accumulate (MAC) operations, which means 16 DP floating-point operation (FLOP) per cycle. The theoretical peak performance of i7-4770K is 249.6 GigaFLOP per second (GFLOPS). Intel Xeon E5-2695V2 is used as server CPU, it has 12 Sandy Bridge cores, where each core provides 8 DP FLOP per cycle, and it has more than three times large on-chip memory (30 MB L3), with a better memory interface. The on-chip memory hiearchy (L1-L3) could create an on-chip memory wall, but in this work I focus on the off-chip memory interface that is the main bottleneck. IBM BlueGene/Q was the state of the art CPU architecture in 2013, which is the building block of power-efficient supercomputing systems. BlueGene/Q has 16 cores with 4 DP MAC per cycle, which results in 204 GFLOPS. E5-2695V2 has 307.2 GFLOPS peak performance at 115W while BlueGene/Q provides 204 GFLOPS at only 55W. Nvidia Tesla K20X represents the family of GPUs. Tesla K20X has 2688 cuda cores operating at 732 MHz, where cuda cores do single precision MAC operations. According to the manufacturer, DP performance of Tesla K20X is 1310 GFLOPS.

The last important class of computing chips is the family of FPGAs. Xilinx Virtex XC7VX1140T was one of the most powerful FPGAs in the case of floating-point multiplications with 3360 DSP slices. The balanced comparison of an FPGA to other processor architectures is challenging. DSP slices of Xilinx Virtex 7 FPGAs perform 25x18

DOI:10.15774/PPKE.ITK.2016.007

bit fixed-point MAC, and every FPGA design is a processor architecture, which has its own computational capability, and memory interface. The maximum operating frequency of a DSP48E1 slice is 741 MHz [R12]. Two connected DSP48E1 slices can perform a single-precision floating-point MAC at the same frequency, thus 2431 SP GFLOPS can be achieved. In the case of DP floating-point, 305 MAC units can be formed at 453 MHz that results in 269.85 DP GFLOPS.

GPUs and FPGAs have more than 1 SP TeraFLOPS theoretical computational power

Table 2.1. Bandwidth limitation of different architectures (2013)

Chip (cores/threads)	Bandwidth GB/s	Memory Type[1]	L2-L3 cache MB
Intel i7-4770K 3.9 GHz (4/8)	25	DDR3 2x1600x64	8
Intel E5-2695V2 3.2 GHz (12/24)	58.3	DDR3 4x1866x64	30
IBM BlueGene/Q 1.6 GHz (16/64)	41.65	DDR3 4x1333x64	32
Nvidia Tesla K20X (2688 cuda cores)	250	GDDR5 6x2662x128	1.5
Xilinx XC7VX1140T (3360 DSP slices)	50	DDR3 4x1600x64	10.42
Chip (cores/threads)	**GFLOPS**	**GFLOPS***	**Caching Multiplier**
Intel i7-4770K 3.9 GHz (4/8)	249.6	3.125	79.87
Intel E5-2695V2 3.2 GHz (12/24)	307.2	7.28	42.19
IBM BlueGene/Q 1.6 GHz (16/64)	204.8	5.2	39.38
Nvidia Tesla K20X (2688 cuda cores)	1310	31.25	41.92
Xilinx XC7VX1140T (3360 DSP slices)	269.85	6.25	86.35

1 : Type (number of channels) x (Mega Transfers / s) x (bits)
GFLOPS* : if 1 MAC (2 FLOP) needs 2x8 byte input from main memory (no cache)

per chip, however, the available off-chip memory bandwidth (25-250 GB/s) can support input only for 3-30 GFLOPS. Caching Multiplier indicates the number of on-chip data reuse that necessary for feeding all PEs continuously. The difference between zero-cache GFLOPS* and the maximum theoretical GFLOPS is 42 times for the GPU and 86 times for the FPGA, and also 42-80 times for the CPUs. It means that input data have to be reused 40-90 times from on-chip memory to reach 100% utilization of PEs.

2.1.2. Processor architectures from 2015

In 2013, all of the processor architectures suffer from memory bandwidth limitation. Two years later the problem is the same as can be seen in Table 2.2. Caching Multiplier can be the measure of memory bandwidth limitation because it shows the difference between computational capacity and the feeding capability of the memory interface. This measure has become worse since 2013 (Fig. 2.1), especially for the FPGA. Intel, Nvidia and

DOI:10.15774/PPKE.ITK.2016.007

Table 2.2. Bandwidth limitation of different architectures (2015)

Chip (cores/threads)	Bandwidth GB/s	Memory Type[1]	L2-L3 cache MB
Intel i7-6700K 4.2 GHz (4/8)	34.1	DDR4 2x2133x64	8
Intel i7-5960X 3.5 GHz (8/16)	68	DDR4 4x2133x64	20
Intel E5-2699v3 3.6 GHz (18/36)	68	DDR4 4x2133x64	45
Nvidia Tesla K80 (4992 cuda cores)	480	GDDR5 12x2500x128	3.25
Xilinx VU13P (12288 DSP slices)	83.3	DDR4 4x2666x64	56.8
Chip (cores/threads)	**GFLOPS**	**GFLOPS***	**Caching Multiplier**
Intel i7-6700K 4.2 GHz (4/8)	268,8	4.25	63.24
Intel i7-5960X 3.5 GHz (8/16)	448	8.5	52.7
Intel E5-2699v3 3.6 GHz (18/36)	504	8.5	59.29
Nvidia Tesla K80 (4992 cuda cores)	2910	60	48.5
Xilinx VU13P (12288 DSP slices)	1850	10.41	177.7

1 : Type (number of channels) x (Mega Transfers / s) x (bits)
GFLOPS* : if 1 MAC (2 FLOP) needs 2x8 byte input from main memory (no cache)

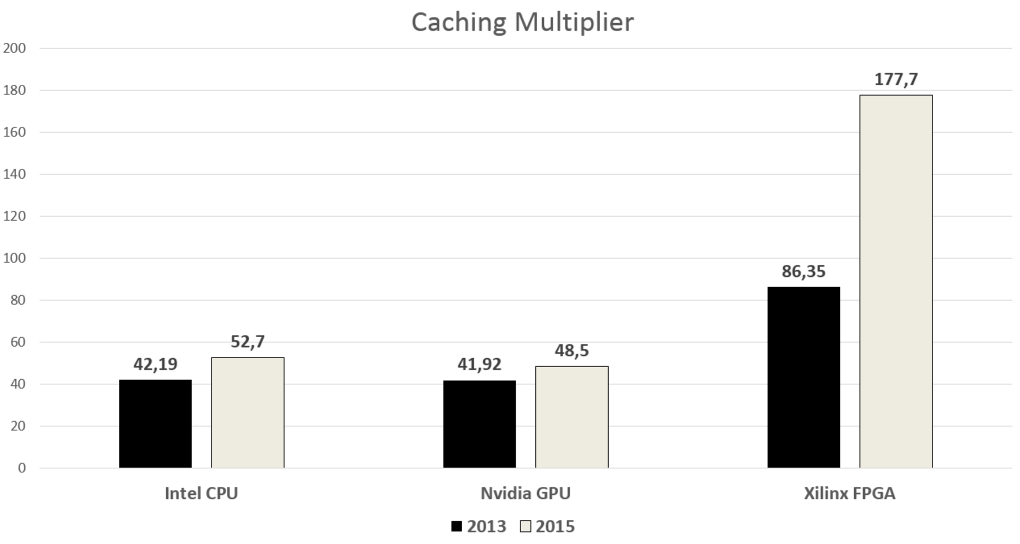

Figure 2.1. Caching Multipliers in 2013 and 2015.

Xilinx (and their competitors) reached great improvement on DP computing capability, while the memory bandwidth improvement was slighter. Nvidia seems to solve the problem (doubled performance with doubled memory bandwidth), but the Tesla K80 is in fact two GK210 GPU with two instances of the GDDR5 interface of Tesla K20X. CPUs and FPGAs strengthen with the new DDR4 interface, but the bandwidth growth is less than the computational capacity growth as can be seen in Fig. (2.1). Xilinx ultrascale FPGAs can also use serial memory interface which consists 64 instances of 15 Gbps channels providing 120 GB/s memory bandwidth. Xilinx improved the DP capability of the FPGAs

DOI:10.15774/PPKE.ITK.2016.007

with the new DSP48E2 slice which performs 27*18 bit multiplications at 891 Mhz thus the DP MAC units require less DSP slices. On the Xilinx VU13P chip, 1755 DP MAC units can be placed with 540 MHz maximum operating frequency. Even with the better 120 GB/s memory interface, more than 100 on-chip data reuse is necessary for the FPGA to reach the 100% utilization.

Processor architectures are bounded by the Speed (GFLOPS)—Power (Watt)—Area (die

Table 2.3. Price and power efficiency of different architectures (2015)

Chip (cores/threads)	Type	Price USD	Power Consumption Watt	DP Efficiency GFLOPS/Watt
Intel i7-6700K 4.2 GHz (4/8)	CPU	350	91	2.94
Intel i7-5960X 3.5 GHz (8/16)	CPU	1000	140	3.2
Intel E5-2699v3 3.6 GHz (18/36)	CPU	3700	145	3.47
Nvidia Tesla K80 (4992 cuda cores)	GPU	5000	2x150	9.7
Xilinx VU13P (12288 DSP slices)	FPGA	20000*	40*	46.25*

*:based on approximations

mm^2 or USD) triangle. Table (2.3) shows the main design aspects of different processor types. High-end GPUs and CPUs utilize the maximum possible power (150 W) with market-driven chip sizes, while high-end FPGAs have 20.000 USD price and reach multiple times better theoretical power efficiency.

Current DRAM (Dynamic Random-Access Memory) technologies are DDR3-DDR4 and GDDR5. These memories are not fully random access because all of them use 8n prefetching, which increases the theoretical memory bandwidth, but also increases the minimal amount of data per transmission. For DDR3, the access granularity is 8x(8-16) bit and 8x32 bit for GDDR5. In GDDR5X the 8n prefetch will be increased to 16n which results in higher theoretical bandwidth, but makes the interface more sensitive to small random memory transactions. DRAMs divide memory into banks, rows, and columns. Two consecutive reads can take a different amount of time. The most efficient way to use a DDR memory is to read a row by burst reads, followed by reading an other row from a different bank, because banks work independently. While a bank closes a row, the other can opens one. With appropriate addressing (bank interleaving), this is a sequential memory read. Any other memory access pattern have less utilized memory bandwidth than the theoretical maximum.

Kilo-processor architectures are memory bandwidth limited, and heavy on-chip data reuse is necessary to provide input for PEs. Furthermore, the memory interface is sensitive to

DOI:10.15774/PPKE.ITK.2016.007

small random accesses. Sequential memory access pattern is needed for maximum utilization of the off-chip memory bandwidth.

2.1.3. Possible Hardware Solutions to the Memory Wall

Better Memory Interface: Higher theoretical memory bandwidth is not enough, the access granularity, and latencies between random accesses are also important. With shorter transmission lines the frequency of the memory interface can be increased. Recent trends try to connect main memory and the processor through 3D via technology [R3, R4], which decreases the wire latencies and makes possible to create wide interfaces with 1024-2048 bits. While latencies drop down and theoretical memory bandwidth increases significantly, the sensitivity of the interface to random access patterns is still high.

Decreased Operating Frequency: Decreased frequency lowers GFLOPS of the processor. With the same memory interface this leads to better efficiency. Computational power linearly depends on the operating frequency, but the power consumption of a processor chip has quadratic frequency dependence. The main indicator of Green Computing GFLOPS/Watt becomes better if the frequency of the processor is decreased.

Increased On-Chip Memory: The rate of on-chip data reuse can be improved with increased on-chip memory. This approach can lead to better PE utilization, but increased on-chip memory needs more chip area, thus less PE can be placed on the same chip, which results in less GFLOPS.

Hardware manufacturers do extensive development to handle memory wall effect, but this is not enough. Software engineers have to use optimized memory access patterns.

2.2. Mesh Computing

Graphs often appear in scientific and industrial computational tasks. The spatial discretization of a 2D surface or a 3D volume can be represented by a graph. These graphs are called meshes and are used in numerical simulations of space-time behavior of sound, heat, elasticity, electrodynamics or fluid flow dynamics. From the design of an airplane to weather estimation, these simulations become the most frequent tasks on supercomputers.

DOI:10.15774/PPKE.ITK.2016.007

If the mesh has a uniform rule-based organization it is called **regular/structured**, otherwise it is **irregular/unstructured**. Irregular meshes make possible the change of mesh density, which is beneficial in many cases. The same numerical precision can be achieved with an irregular mesh with much fewer elements than a regular mesh requires. However, computations on irregular meshes are more complex.

In mesh computing, the following questions arise: How to generate the mesh? (regular/irregular, number of elements); How to partition a mesh for parallel computing? (optimization goals, partitioning method, number of parts); How to organize the corresponding data in main memory? In this work I deal with the second and third questions, furthermore, I want to show that these two questions are in fact one.

In the later sections, I use the task of explicit numerical approximation of a system of

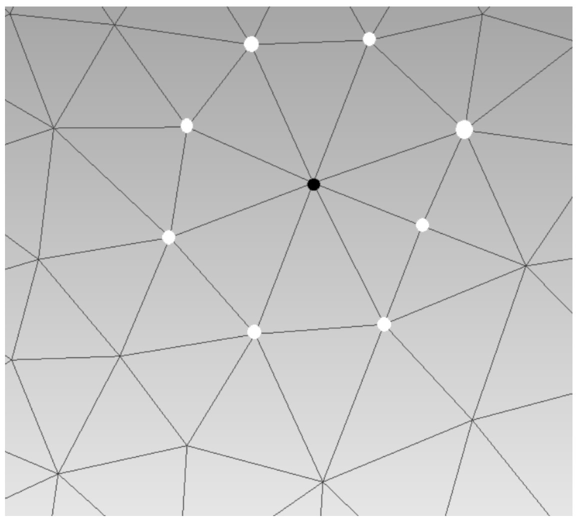

Figure 2.2. 2D vertex-centered mesh example. The black node depends on the white neighboring nodes.

partial differential equations (PDE) where the spatial discretization is defined by a mesh. The discretization is called **vertex-centered** if the represented physical variable is defined at the vertices. In the case of **cell-centered** discretization the represented physical variable is defined at the center of the primitive element (triangle, tetrahedron, etc.). The **discretization stencil** defines which neighboring elements and how contributes to the dynamics of each element. The simplest discretization stencil contains the directly connected neighbors (Fig. 2.2), which has one distance from the current node in the graph.

DOI:10.15774/PPKE.ITK.2016.007

2.3. Dataflow Computing

In dataflow computing, an operation is executed immediately if its operands are available. In the case of classic von Neumann computing, an operation is executed when the serial control reaches the operation. The result of a dataflow operation can invoke multiple operations which can be executed in parallel. Dataflow computers are typically pipelined architectures in which a continuous data stream goes through an acyclic graph structure of operators.
Neumann machines often support SIMD (Single Instruction Multiple Data) computing to utilize multiple processing elements in parallel. Dataflow machines could provide the same extension with multiple identical pipes. This technique increases the input and output bandwidth requirements of the architecture, but gives linear speedup.

2.3.1. Dataflow computing on a mesh

As I mentioned in the previous section, in this work an explicit PDE solver is used as a test problem. In each time step, the state variables of all discrete mesh elements have to be updated. For the update, all the constants and state variables of the given node and its neighborhood are required. Mesh data have to go through the dataflow processor unit which can update a finite number of nodes at the same time. The necessary input has to be loaded on-chip for a node update, thus, all the required neighboring nodes have to be stored. The processor has finite memory resources which make a limit on the maximum distance of dependent nodes in the input stream, otherwise, the mesh data need to be loaded multiple times, and we lose most of the benefits which come from the dataflow concept.
Fig. (2.3) shows the memory transfers in the case of a dataflow machine that works on

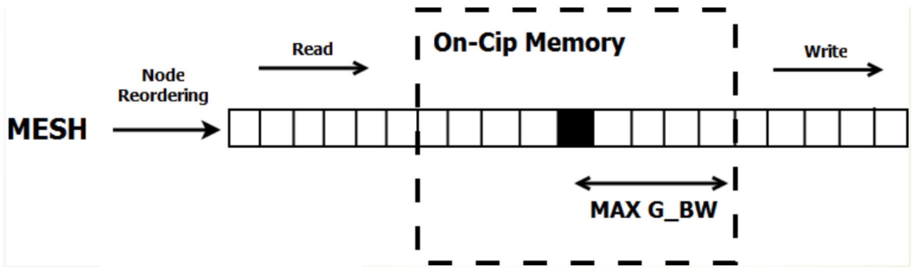

Figure 2.3. Linearized mesh data goes through on-chip memory buffer.

mesh data. First, the mesh elements have to be ordered into a serial stream, which goes

DOI:10.15774/PPKE.ITK.2016.007

through an on-chip memory buffer, from which the dataflow arithmetic can access the required data for a node update. Ordering is a key step for dataflow computing because the ordering defines the maximum distance of dependent elements in the stream. If we see the adjacency matrix of the graph, this distance is equal to the largest distance of nonzeros in a row of the matrix. If we consider always updating the middle element of the on-chip memory buffer, this distance is two times the largest distance of a nonzero from the main diagonal, which distance is called graph or matrix bandwidth (G_BW). The size of on-chip memory buffer gives an upper bound on possible graph bandwidth.
An ordering is required which results in less G_BW than a bound. There are effective matrix bandwidth minimization heuristics, however, to find an optimal ordering is NP-complete. In Fig. (2.4) the adjacency matrix is shown for the same graph before and after reordering. Reordering heuristics like Cuthill-McKee or GPS are effective and fast, but it is possible that a graph can not be reordered under a given G_BW bound. In later chapters, I will give some possible solution to this problem.

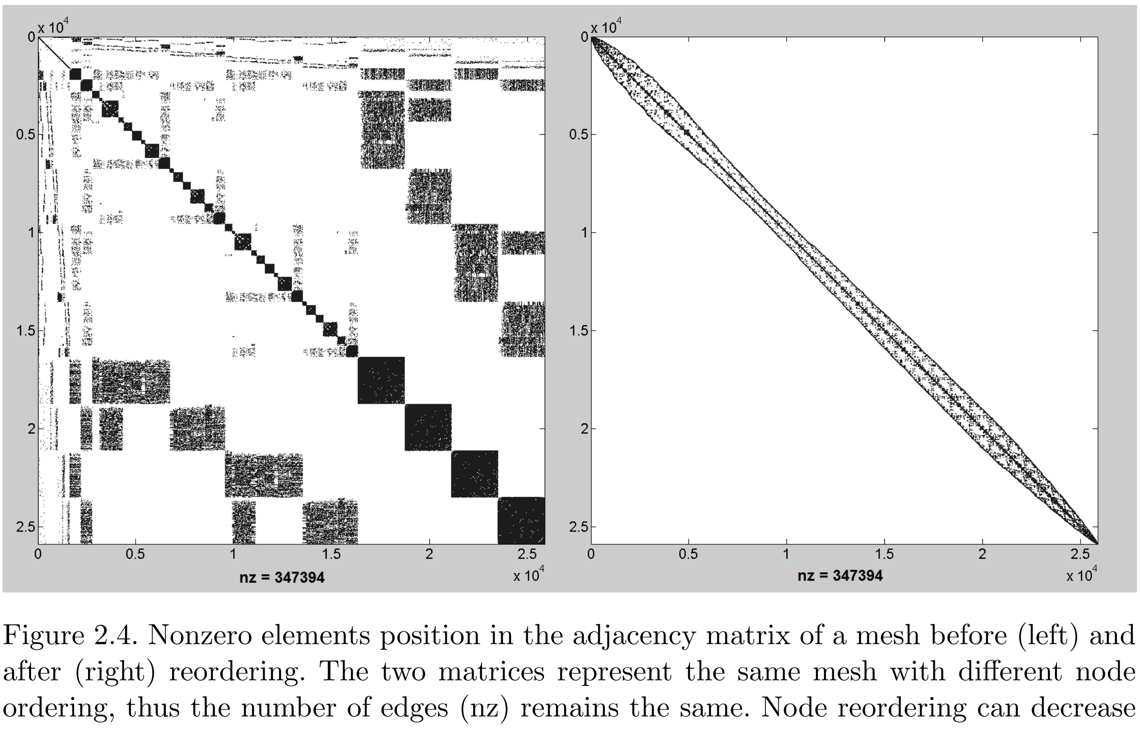

Figure 2.4. Nonzero elements position in the adjacency matrix of a mesh before (left) and after (right) reordering. The two matrices represent the same mesh with different node ordering, thus the number of edges (nz) remains the same. Node reordering can decrease the graph bandwidth significantly.

DOI:10.15774/PPKE.ITK.2016.007

2.3.2. Memory access optimization and interprocessor communication

Optimized memory access is essential for high processor efficiency. Current mesh partitioning methods focus only on inter-processor communication, however, processor-memory communication is also critical for recent processor architectures. This contribution suggests considering the properties of inter-processor and processor-memory interfaces with the processor's caching capability in mesh partitioning.

Dataflow Computing is a possible solution for memory wall effect because it requires a continuous input data stream. Streams have fully sequential memory access pattern, which means optimal memory bandwidth efficiency. Recently, Dataflow Machines are introduced for explicit PDE computations on structured [R13] and unstructured [J1] meshes, and they are proved to be much faster than any other architecture.

The benefits of these architectures can be exploited only if the input stream has optimized data locality. A segment of the input stream can be cached in the on-chip memory. However, all data dependencies have to be inside a segment. The size of the available on-chip memory defines an upper bound (BW_Bound) on the maximum distance of dependent data in the stream. This is a hard constraint. If dependent data have greater distance in the stream than the bound, DM can not handle it.

Inter-processor communication is also important for Dataflow Machines. If the processors have to wait for data from the inter-processor communication channel, the efficiency also becomes lower. To evade this effect, Eq. (2.1) must be considered, which defines a bound ($COMM\_Bound$) on the ratio of inter-processor communication.

$$\frac{\text{communication [byte]}}{\text{input for computation [byte]}} \leq \frac{\text{inter-processor bandwidth [byte/s]}}{\text{off-chip memory bandwidth [byte/s]}} \tag{2.1}$$

$COMM\_Bound$ is defined by the ratio of the inter-processor and off-chip DRAM bandwidth. In [J1] Alpha-Data ADM-XRC-6T1 cards are used which provide 12.5 GB/s theoretical off-chip memory bandwidth, and 1.25 GB/s bandwidth between the cards, thus $COMM\_Bound = 0.1$. In [R13] two custom PCI Express add-on cards are presented. The first has 9.375 GB/s off-chip memory bandwidth with 2.5 GB/s inter-processor bandwidth, which results $COMM\_Bound = 0.26$. The second configuration has 37.5 GB/s off-chip memory bandwidth with 5 GB/s inter-processor bandwidth, thus $COMM\_Bound = 0.13$.

DOI:10.15774/PPKE.ITK.2016.007

2.3.3. Data Locality in Mesh Computing

A graph G can be associated to each mesh, by converting each mesh element to a vertex, and each face to an edge. In the case of an explicit PDE solver, data locality is the maximum distance between adjacent nodes in the linearized stream of mesh data. This distance is proportional to $B_f(G)$, which is is the graph bandwidth of G according to node ordering f (details in Sec. 4.3). Data dependencies in the numerical method are described by the discretization stencil. When the discretization stencil includes only the adjacent elements, its width is $s = 3$. With these notations, the maximum distance of dependent nodes is $(s-1) \cdot B_f(G) + 1$. Multiple explicit iterations can be computed with one off-chip read if the intermediate results are also stored in an on-chip buffer, and the dataflow arithmetic units are connected in a pipeline [J1]. The relation between the graph bandwidth and the BW_Bound is given in Eq. (2.2).

$$\text{Iterations} \cdot \{(s-1) \cdot B_f(G) + 1\} \leq BW\_Bound \tag{2.2}$$

BW_Bound is determined by the maximum size of available on-chip memory. Data Locality bounds are 30-90K and 40-300K for architectures described in [J1], and [R13], respectively.
Minimization of the graph bandwidth can provide data streams with feasible data locality. The goal of minimization is to find an ordering f, for which the graph bandwidth is minimal. The achievable minimal graph bandwidth depends on the graph. A partitioning method defines subgraphs, which affect the achievable graph bandwidth. This effect is investigated in Section 4.3.

Data locality (BW_Bound) and inter-processor communication ($COMM\_Bound$) have to be considered together in mesh partitioning. In the following chapter, an overview is given on existing dataflow architectures which can utilize my results on data locality based mesh partitioning.

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

3. Chapter

Dataflow Machines

This chapter gives an overview on existing dataflow machine architectures of different application areas. The case of mesh computing is presented in more details through two special-purpose dataflow machines.

3.1. FPGA and All Programmable System on Chip (APSoC) architectures

Dataflow machines require high hardware flexibility which can be only achieved by ASIC or FPGA chips. Before I introduce DM applications, I give a brief overview of FPGA and APSoC architectures through Xilinx products.

3.1.1. Field Programmable Gate Array (FPGA)

The evolution of FPGA chips has started from real logic gate arrays (Field Programmable Logic Array—FPLA) and has shifted towards more complex building blocks. The current FPGA technology is grounded on the LCA (Logic Cell Array) architecture which is introduced by Xilinx in 1985 [R14]. This minimal design has a grid of logic cells which is surrounded by Input/Output Blocks (IOB). The LCA has a programmable interconnect between all elements. Each logic cell consists a logic function generator and 1-bit storage (flip-flop).

Later, more complex and special building blocks appear in the Virtex architecture (Fig. 3.1). Logic cells evolved to Configurable Logic Blocks (CLB) that consists 4 Lok-Up Tables (LUT), 4 Carry generators and 4 flip-flops as it can be seen in Fig. 3.2. CLBs form

DOI:10.15774/PPKE.ITK.2016.007

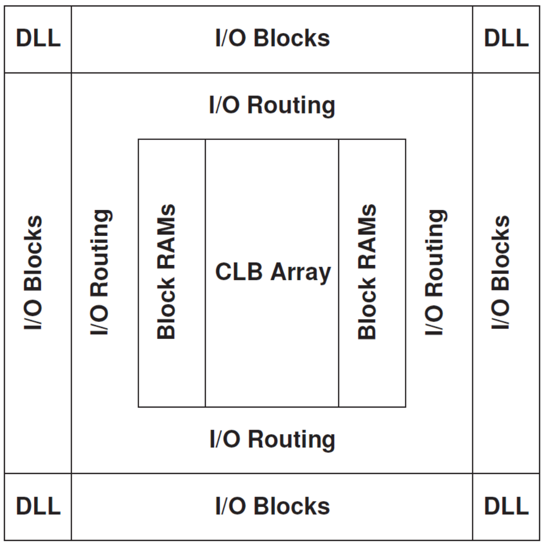

Figure 3.1. The Virtex Architecture ([R14]).

a grid, which is surrounded by IOBs. IOBs are connected to the CLB matrix through special programmable interconnect (I/O Routing) while CLB-CLB connections are provided by the General Routing Matrix (GRM). The new Delayed Locked Loop (DLL) blocks are responsible for clock handling, and dedicated memory units are also added to the dsign (Block RAMs). Each BRAM module is a 4 Kbit dual-port RAM with independent control signals and configurable data width. LUTs in CLBs are not only function generators, but they can also be used as RAMs or shift registers. CLBs can perform full-adder logic and multiplexing.

The key ability of FPGAs is the programmable hardware connections. Fig. (3.3) shows

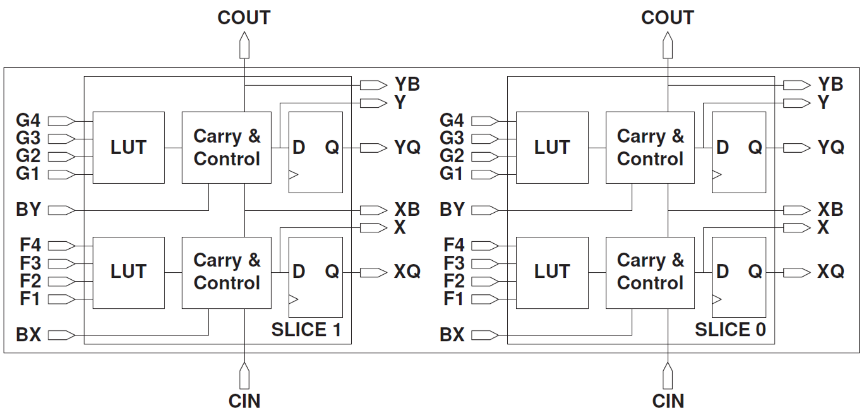

Figure 3.2. 2-Slice Virtex CLB ([R14]).

DOI:10.15774/PPKE.ITK.2016.007

the direct connections between neighboring CLBs and GRM crosses. Each programmable wire cross adds a delay to the signal path, thus long range and mid range lines are also added to the design for routing long-distance paths.

In later generations, the size of LUTs and BRAMs increased, and new blocks appeared

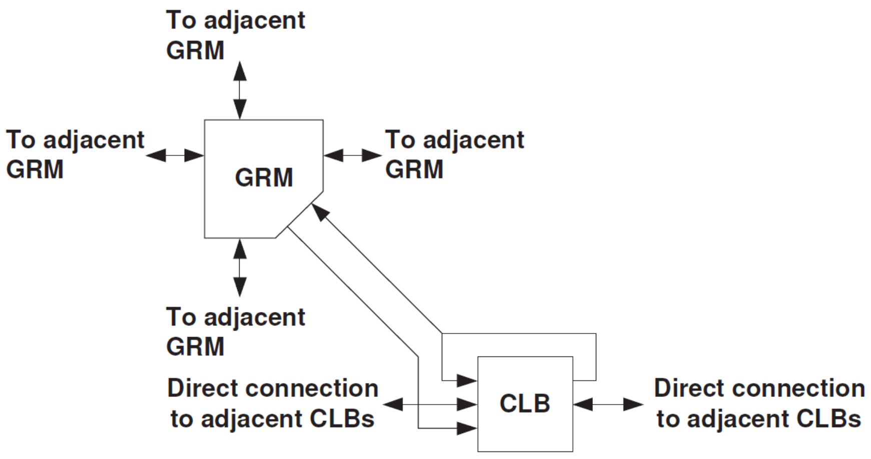
To adjacent GRM
To adjacent GRM
GRM
To adjacent GRM
To adjacent GRM
Direct connection to adjacent CLBs
CLB
Direct connection to adjacent CLBs

Figure 3.3. Local interconnects in a Virtex FPGA. ([R14]).

such as DSP slices and dedicated Ultra RAMs. Based on the most common applications, more and more functionalities got dedicated support on the chip. For instance, in ultra-csale FPGAs the BRAMs have dedicated cascade support as it can be seen in Fig. (3.4). Because of the demand for larger on-chip memory, Xilinx added a new type of memory resource which is called Ultra RAM. Ultra RAMs have less reconfigurability than Block RAMs, but they are perfect for larger on-chip memory formation. Ultra RAM has 288 Kbit memory in a single block and dedicated cascade support as BRAMs.

The most important module in the case of High-Performance Computing (HPC) is the DSP slice because it can perform multiplications. Fig. (3.5) shows the DSP48E2 slice of the ultrascale family. As I mentioned in the previous chapter, this module performs 27*18 bit multiplications at 891 Mhz. One of the benefits of FPGAs is custom precision computing, however, the need for standard double precision in real applications forces Xilinx to increase the bit width of the multiplier.

DOI:10.15774/PPKE.ITK.2016.007

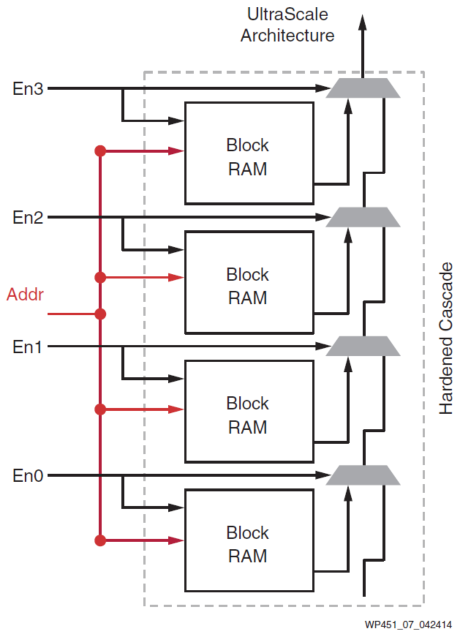

Figure 3.4. Dedicated Block RAM Cascade in UltraScale Architecture ([R15]).

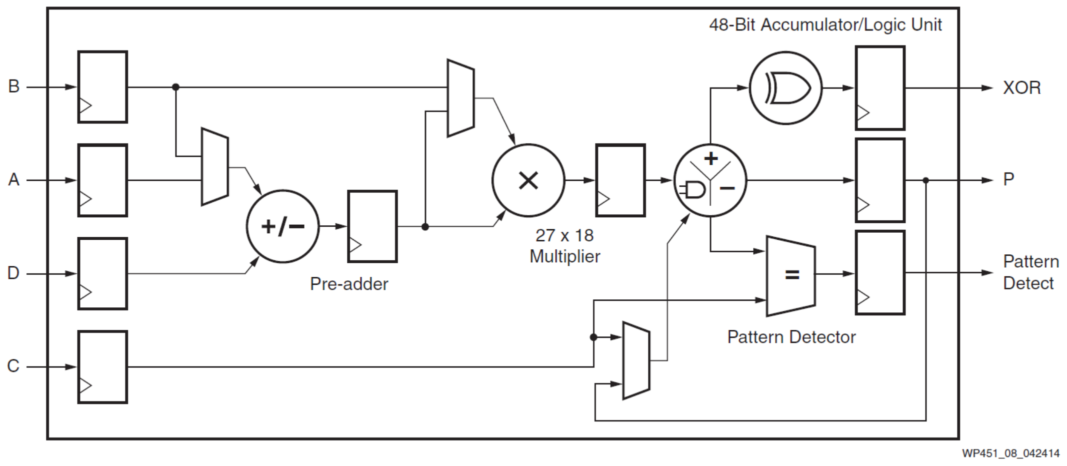

Figure 3.5. Enhanced DSP in UltraScale Architecture ([R15]).

3.1.2. All Programmable System on Chip (APSoC)

Complex applications require different types of computing functionalities. Hybrid architectures are developed to handle these challenges. Here I show the Zynq 7000 APSoC chip (Fig. 3.6) which consists a dual-core ARM Cortex-A9 processor (Processing System—PS) and a 7th series Xilinx FPGA part (Programmable Logic—PL).

DOI:10.15774/PPKE.ITK.2016.007

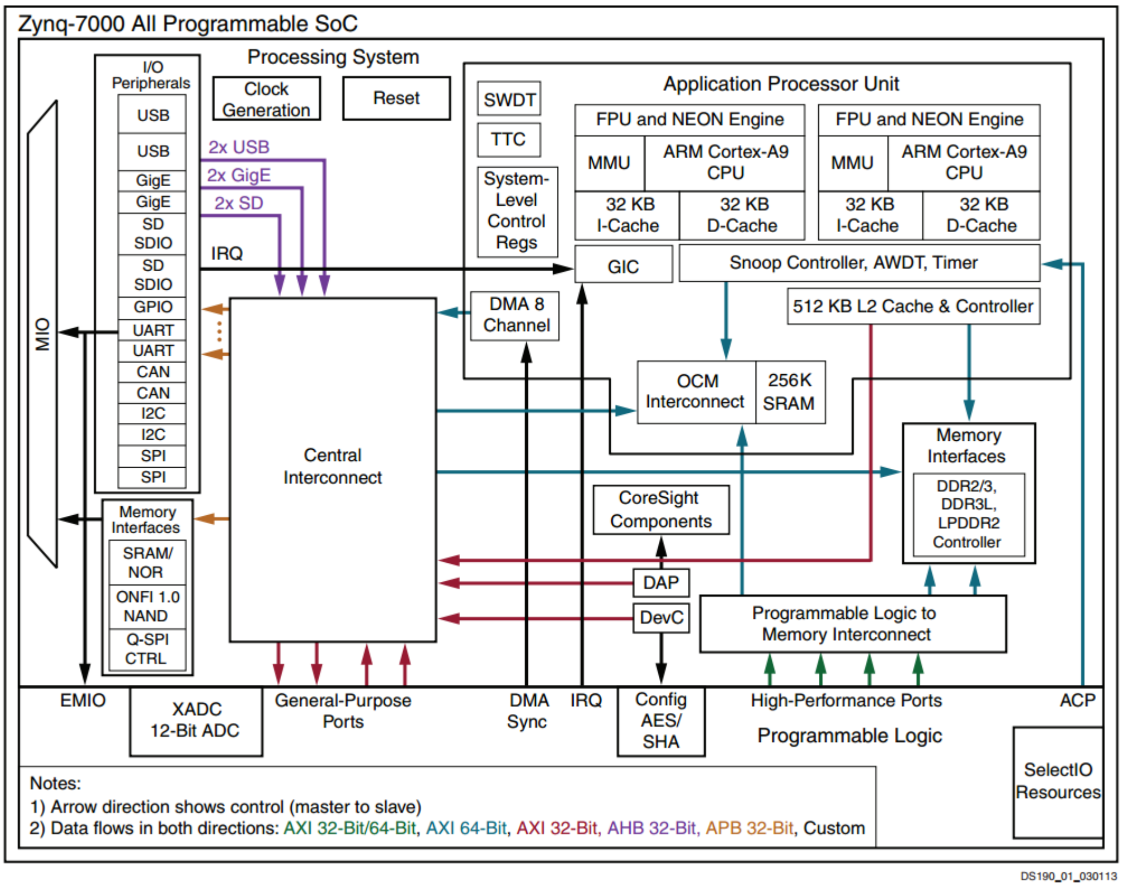

Figure 3.6. Zynq-7000 All Programmable SoC Overview ([R16]).

The PS side of the chip can replace the host CPU and can communicate with the FPGA part on-chip. The PS side can be programmed in C, and the Vivado toolchain generates the necessary drivers for the custom logic on the FPGA side which makes the APSoC easy to use. The benefits of custom FPGA cores and a standard ARM CPU are joined. 32/64 bit AXI4 interfaces to connect the different types of custom PL modules and the PS. As shown in Figure (3.7), these interfaces connect the PL to the memory interconnect via a FIFO controller. Two of the three output ports go to the DDR memory controller, and the third goes to the dual-ported on-chip memory (OCM).

The PS side runs at maximum 1 GHz while the FPGA part maximum frequency is based on the application (250 MHz for Zynq PS AXI interface). If the application can utilize the parallel computing capabilities of the PL side the result is a power-efficient high-performance design.

DOI:10.15774/PPKE.ITK.2016.007

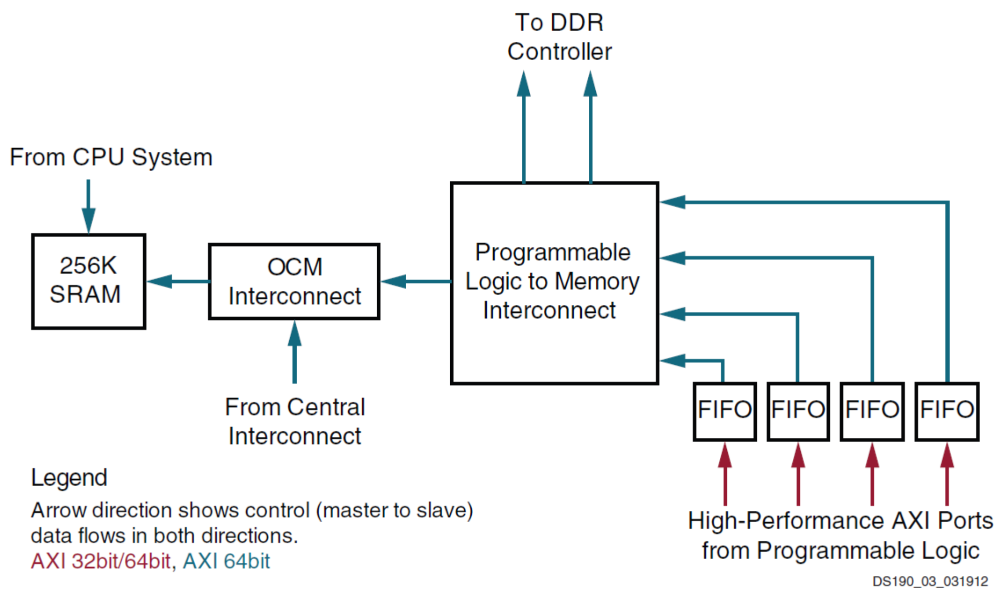
To DDR Controller
From CPU System
256K SRAM
OCM Interconnect
Programmable Logic to Memory Interconnect
From Central Interconnect
FIFO
FIFO
FIFO
FIFO
Legend
Arrow direction shows control (master to slave) data flows in both directions.
AXI 32bit/64bit, AXI 64bit
High-Performance AXI Ports from Programmable Logic
DS190_03_031912

Figure 3.7. PL Interface to PS Memory Subsystem ([R16]).

3.2. Existing hardware solutions of DMs

The following examples represent the common application areas and hardware solutions of dataflow machines. Except the NeuFlow ASIC implementation, all of these architectures are realized on FPGA chips.

3.2.1. Maxeler accelerator architecture

This architecture could be the general framework example. The DM is placed on an FPGA-based accelerator board which is connected to a general purpose CPU host through PCI Express.

The application kernel is transformed automatically from a dataflow graph into a pipelined FPGA architecture, which can utilize a large amount of the parallel computing resources on the FPGA chip. The host application manages the interaction with the FPGA accelerators while the kernels implement the arithmetic and logic computations in the algorithm. The manager orchestrates data flow on the FPGA between kernels and to/from external interfaces such as PCI Express. In [R17] this architecture is used for resonance-based imaging in a geoscientific application which searches for new oilfields. The implementation involves 4 MAX3 FPGA accelerator cards. Each card has a lar-ge Xilinx Virtex-6 FPGA and is connected to the other FPGAs via a MaxRing connection.

DOI:10.15774/PPKE.ITK.2016.007

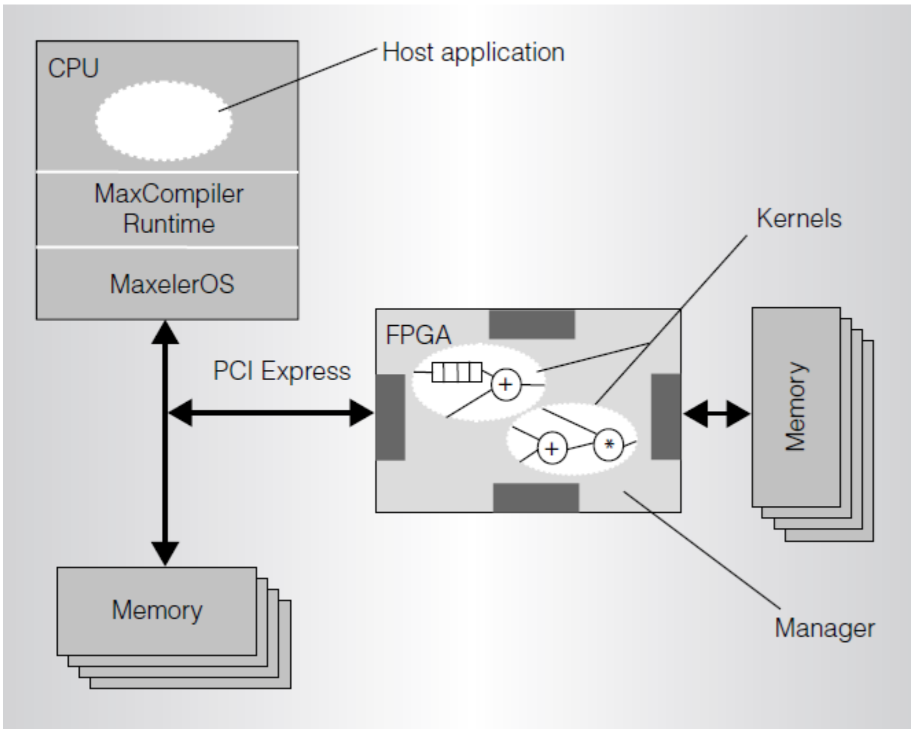

Figure 3.8. Maxeler accelerator architecture ([R17]).

3.2.2. HC1 coprocessor board

In paper [R18], the authors present an accelerator board which is made for the investigation of evolutionary relations of different species. The computational problem includes a maximum likelihood-based phylogenetic interface with the Felsenstein cut method. BEAGLE is a programming library which contains phylogenetic algorithm implementations for many architectures. This library is also extended to FPGA platforms which name is Convey HC-1.

The corresponding hardware solution is based on a Xeon server CPU host with 24 GB memory. The accelerator includes 4 Virtex-5 FPGAs, which can access 16 GB on-board memory through a full crossbar network (Fig. 3.9). The FPGAs have a ring topology inter-FPGA communication network. When the input problem is distributed among the FPGAs, the topology has to be considered, because the communication between neighbors is multiple times cheaper than the communication of 2 FPGA-s which are not adjacent. The large on-board memory makes possible to ignore the relatively slow PCI Express interface during the computation.

DOI:10.15774/PPKE.ITK.2016.007

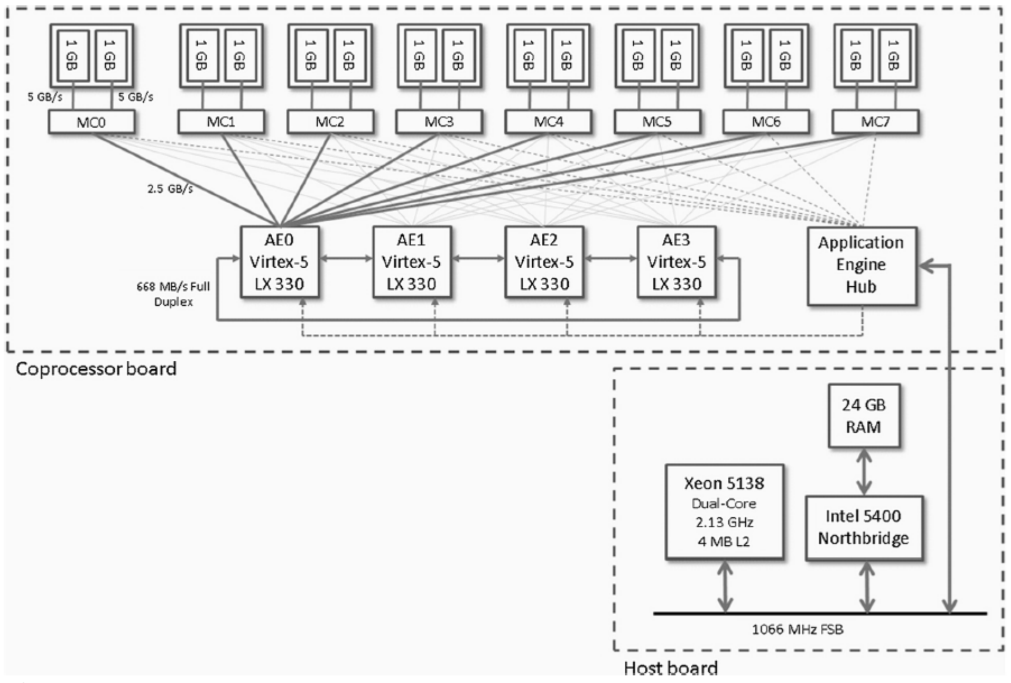

Figure 3.9. The HC1 coprocessor board. Four application engines connect to eight memory controllers through a full crossbar ([R18]).

3.2.3. Multi-Banked Local Memory with Streaming DMA

In the project that is shown in [R19] a special on-chip memory organization is used. The multi-way parallel access memory is a perfect solution to feed the dataflow arithmetic. The on-chip memory is filled by a streaming DMA which reads the off-chip memory continuously. This DMA strategy utilizes the whole off-chip DRAM bandwidth which is the limiting factor in many applications.

Figure (3.10) shows the organization and connections of an Application-Specific Vector Processor (ASVP). Each ASVP has a simple scalar processor (sCPU) for scheduling the vector instructions (α), for programming the streaming DMA engine (γ) and for optional synchronization with other ASVPs through Communications Backplane (δ). The vector instructions are performed by the Vector processing Unit (VPU) which can access the BRAM-based Local Storage banks in parallel (β). The maximal operating frequencies of the VPU are 166MHz, 200MHz, and 125MHz, for Virtex 5 (XC5VLX110T-1), Virtex 6 (XC6VLX240T-1), and Spartan 6 (XC6SLX45T-3) FPGAs, respectively.

Multiple ASVPs can be connected to the streaming memory interface, if there is enough resource on the FPGA and there is enough off-chip memory bandwidth. For different app-

DOI:10.15774/PPKE.ITK.2016.007

lications, only the Vector Processing Unit has to be changed, most of the architecture can be unchanged which saves development cost.

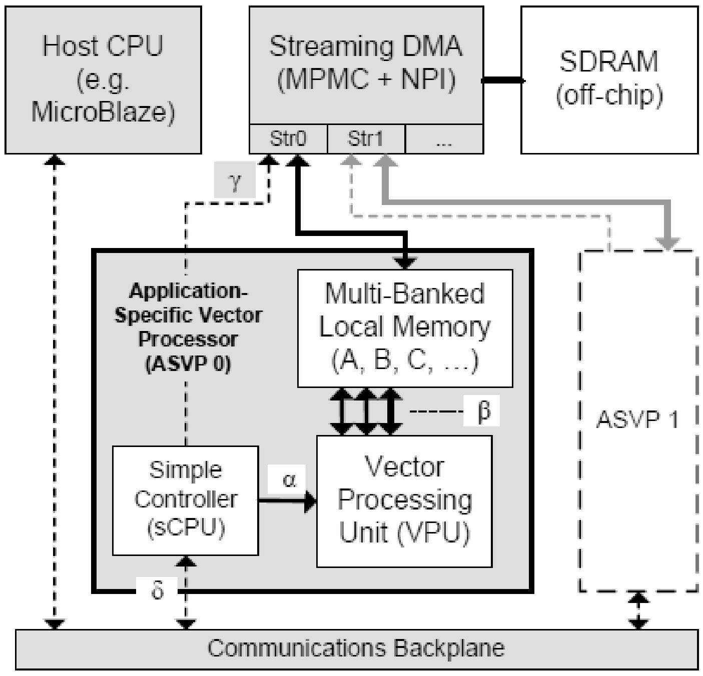
Host CPU (e.g. MicroBlaze)
Streaming DMA (MPMC + NPI)
Str0
Str1
...
SDRAM (off-chip)
γ
Application-Specific Vector Processor (ASVP 0)
Multi-Banked Local Memory (A, B, C, ...)
β
ASVP 1
Simple Controller (sCPU)
α
Vector Processing Unit (VPU)
δ
Communications Backplane

Figure 3.10. A system-level organization of an Application-Specific Vector Processor core ([R19]).

3.2.4. Large-Scale FPGA-based Convolutional Networks

An important application area is the 2D or higher dimensional convolutions. These computational tasks appear in almost all image or video processing applications, and they are computationally expensive. In Fig. (3.11) the architecture of [R21, R20] is shown when it is configured for a complex image processing task. The processor is formed by a 2D matrix of processor blocks. Each block has 6 predefined computing module with independent in/out interfaces which can be connected optionally through a connection matrix. In the given example, the 3 upper blocks perform a 3x3 convolution while the middle 3 block perform another 3x3 convolution. The two results are added by the left down block, and then the down center block computes a function.
In Fig. (3.12) can be seen the manufactured chip layout. It is interesting that the streaming part is as large as the computing part on the chip. The flow CPU is used for programming

DOI:10.15774/PPKE.ITK.2016.007

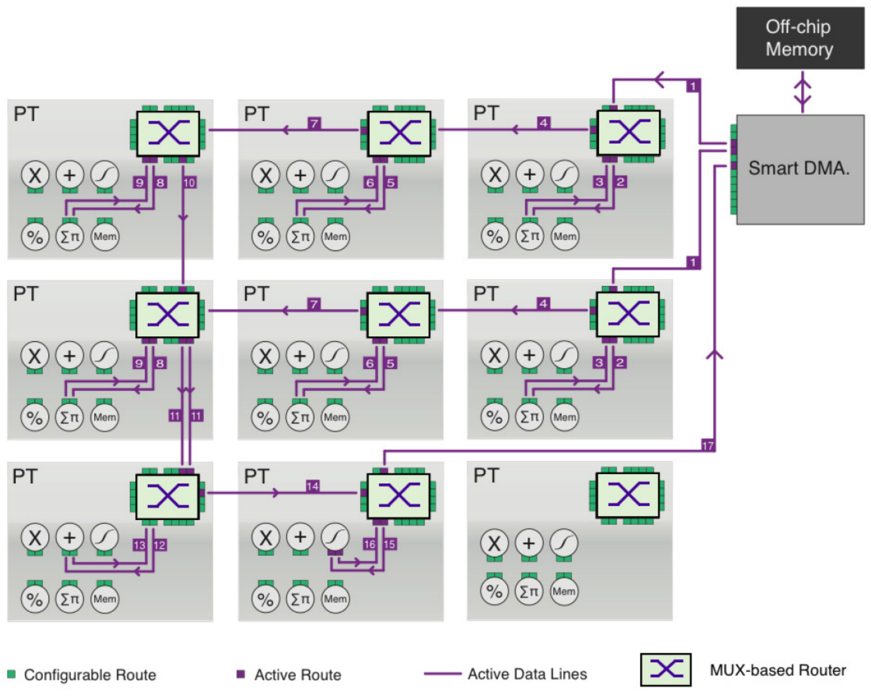

Figure 3.11. NeuFlow application example ([R20]).

the other parts and makes possible fast reconfigurations during the computation.

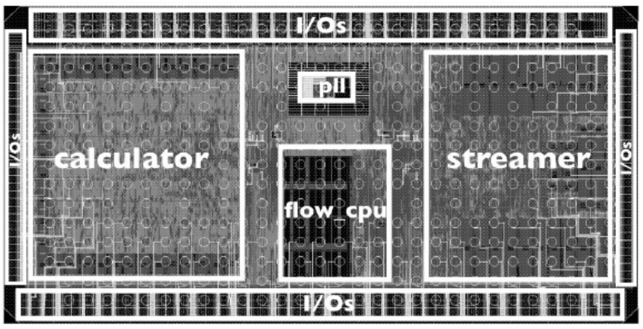

Figure 3.12. Chip layout in a $2.5 \times 5\ mm^2$ die area ([R20]).

3.2.5. Pipelined Maxeler Accelerators

This architecture is based on the one mentioned in [R17]. Fig. (3.13) show the Maxeler MPC-C architecture and the corresponding design flow. The usage of dataflow machines becomes much easier with the projects like Maxeler, which provides frameworks which requires only C-like programming skills, and generates the hardware description codes automatically. In the paper [R22] this architecture is used for electromagnetic field simu-

DOI:10.15774/PPKE.ITK.2016.007

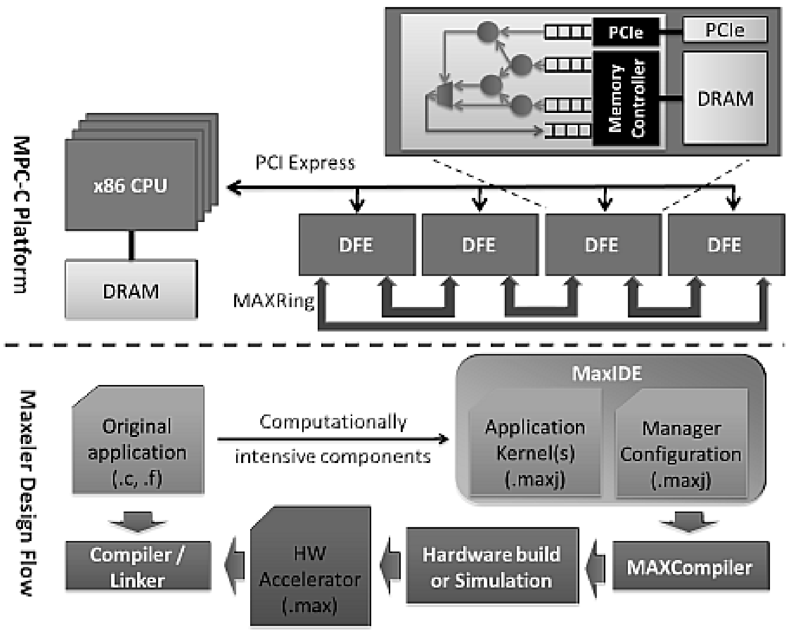

Figure 3.13. MPC-C platform architecture and Maxeler design flow ([R22]).

lations. The host consists general purpose CPUs (two Intel Xeon X5650 2.7GHz 6-core CPUs), which communicate with FPGA-based boards (four MAX3 DFE cards) through PCI Express. The FPGAs has their DRAM, and they are connected in a ring topology.

Here I want to show the possibility of deep pipelining. In the case of an iterative method, the operations of one iteration can be copied after each other or with timesharing and data back feeding multiple iterations can be computed without off-chip memory transfers. The Figure (3.14) shows the possible pipelining depths. The electric (E) and magnetic (H) fields can be computed in two steps on the same processor unit (a). E and H can be computed in a pipeline which means two times speedup with two processor units (b), and if there are enough resources, more iterations can be performed at once with deep pipelining (c).

DOI:10.15774/PPKE.ITK.2016.007

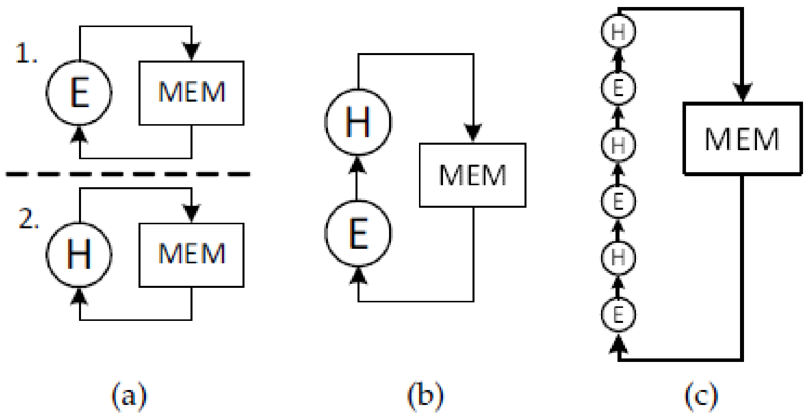

Figure 3.14. Possible pipelined approaches. (a) no pileline (b) single iteration (c) multiple iterations ([R22]).

3.3. Off-chip memory streaming techniques

This section gives an overview of the three most common streaming techniques between main memory and the dataflow processor unit. In [R23] the authors did experiments based on the Himeno benchmark which is frequently used in performance evaluation. The method is named after Dr. Ryutaro Himeno and includes the Jacobi iteration based solution of the Poisson equation which is part of the Navier-Stokes equations.

For the investigations, the MAX3 acceleration card was used which has a Virtex-6 SX475T

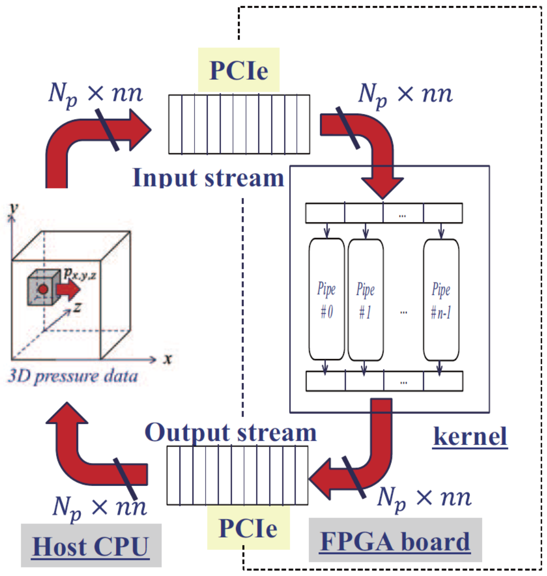

Figure 3.15. Direct feed from host main memory through PCI-Express ([R23]).

DOI:10.15774/PPKE.ITK.2016.007

FPGA, 24GB DDR3 memory and PCI express gen2 x8 interface. On the FPGA multiple dataflow solver units (pipe) can be implemented. The number of pipes is limited by the logic resources of the FPGA, however, these pipes require high memory bandwidth.
In the benchmark problem N_p denotes the number of sample points in the 3D spatial domain, where the pressure (p) must be determined in each iteration. The number of iterations is nn thus $N_p \times nn$ data elements must be communicated during the computation. The tests are done for 3 different sized problems: S 65x65x129 (2.1 MB), M 129x129x257 (16.3 MB) and L 257x257x513 (129.3 MB). The clock speed of FPGA designs is set to 100 MHz.
The first possible way of streaming is the direct feed from the main memory of the host through the PCI-Express bus as we can see in Fig. (3.15). In this case, only 8 pipes can be supplied because the PCI Express memory bandwidth limits the performance at 8.33 GFLOPS.
The other extreme case when the whole problem is placed in an on-chip memory buffer.

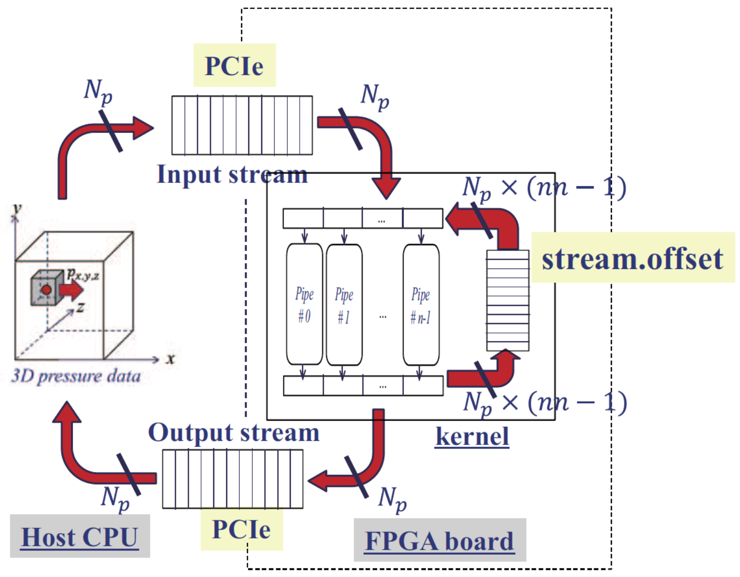

Figure 3.16. Input loaded to on-chip local memory and processors feeded from on-chip memory ([R23]).

It is viable only for small problems (S data set) because the on-chip memory need is relative to the size of the problem, but this case shows the maximal performance available. Figure. (3.16) shows the block diagram and figure (3.17) the measurement results, which indicates 145 GFLOPS at only 100 Mhz with 48 parallel pipes. The design with 48 pipes could run at maximum 110 MHz which results in 155 GFLOPS peak performance.

DOI:10.15774/PPKE.ITK.2016.007

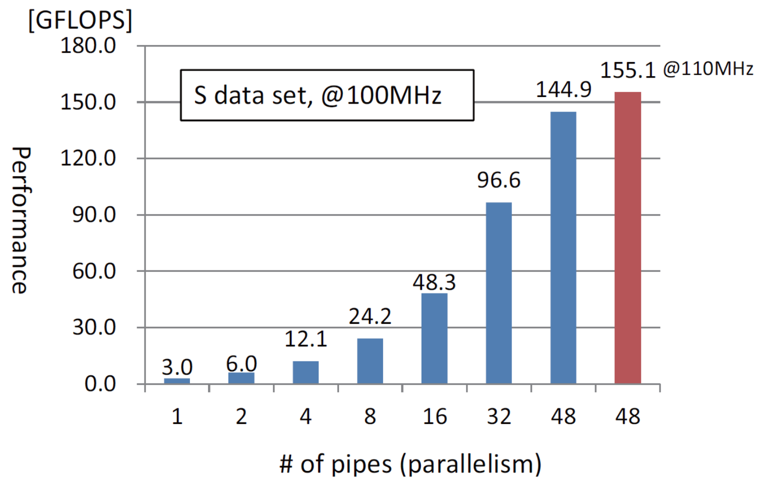

Figure 3.17. Results of on-chip buffer feeding ([R23]).

The on-board 24 GB DDR memory can handle relative large problems and can also exc-

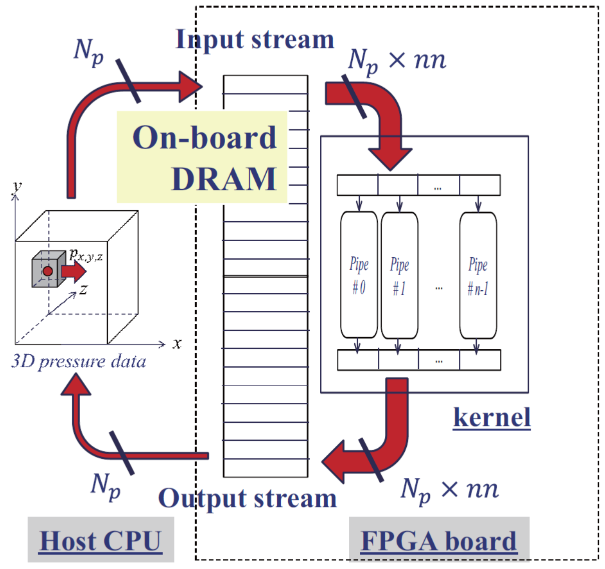

Figure 3.18. Input loaded to on-board DRAM and processors feeded from on-board memory through off-chip memory interface ([R23]).

lude the slow PCI Express bus (Fig. (3.18)). In the beginning, the input data is loaded to the on-board DDR and after the whole computation, the final result is sent back to the host. In this case, the FPGA has to include memory address generators which consume resources thus only 32 pipes can be implemented. The peak performance is 97,6 GFLOPS (Fig. (3.19)), however, this approach is applicable for large problems as well. In high-performance computing, this technique is the most common to feed dataflow processors.

DOI:10.15774/PPKE.ITK.2016.007

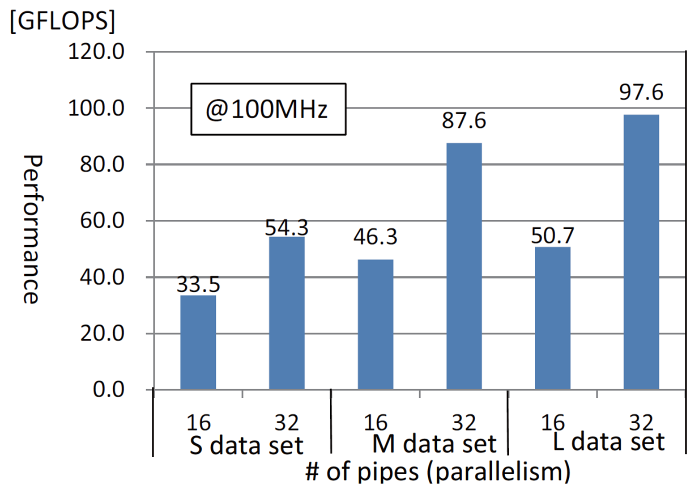

Figure 3.19. Results of the on-board memory feeding ([R23]).

In the following section, two special-purpose mesh computing dataflow machines are introduced, and both of them use the on-board DDR memory streaming.

3.4. Special-Purpose DMs for mesh computing

The following two architectures are specialized for mesh computing. Both of them have found to be the best architectures in the case of explicit PDE computing, because of the total off-chip memory bandwidth utilization [R13, J1]. The previous chapter has given an introduction to these special DMs.

3.4.1. DM for structured meshes

The Maxeler framework has been mentioned multiple times in this chapter. Here I show the application of [R13] and focus on the case of distributed mesh computation on multiple DMs. The corresponding hardware solution includes 4-16 FPGA-based accelerators which are connected to the host through PCI Express and has a ring topology interconnection network. Each FPGA (Dataflow chip) has its on-board memory as can be seen in Fig. (3.20).

The input is a 3D structured discretization of a rectangular space domain. Fig. (3.21) shows the distribution of the domain among the DMs. The domain has been cut according to one dimension into equal sized pieces. This distribution is ideal for the ring topology, furthermore, if the DMs are synchronized, they can share the boundary cell information

DOI:10.15774/PPKE.ITK.2016.007

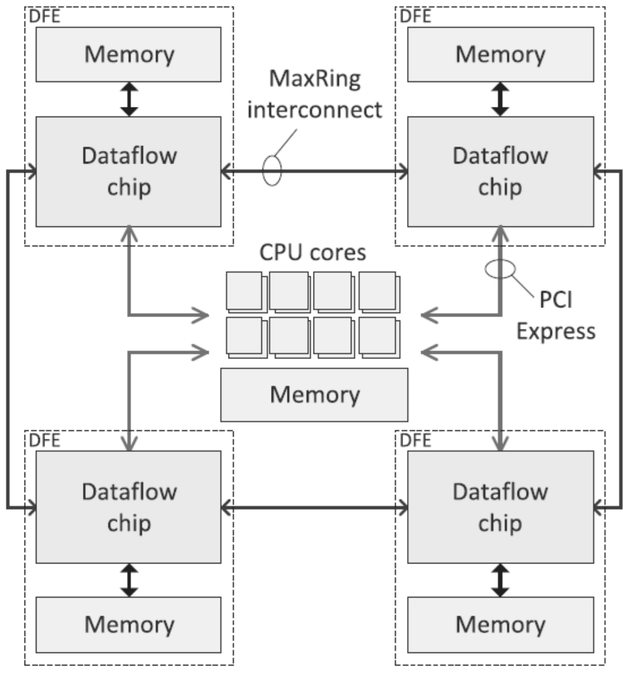

Figure 3.20. Architecture of a compute node. Each of the Data-Flow Engines (DFE) is connected to the CPUs via PCI Express and has a high bandwidth MaxRing interconnection to its neighbors ([R13]).

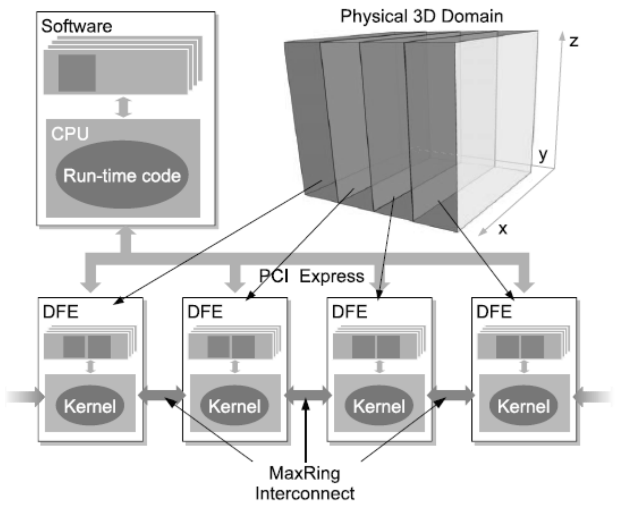

Figure 3.21. One-dimmensional decomposition of the problem domain to parallelize across multiple DFEs linked with MaxRing ([R13]).

with each other without extra off-chip memory transfers. This architecture also has a data locality limit which comes from the available memory resources on the FPGA chip, as it has been mentioned in Sec. (2.3).

DOI:10.15774/PPKE.ITK.2016.007

3.4.2. DM for unstructured meshes

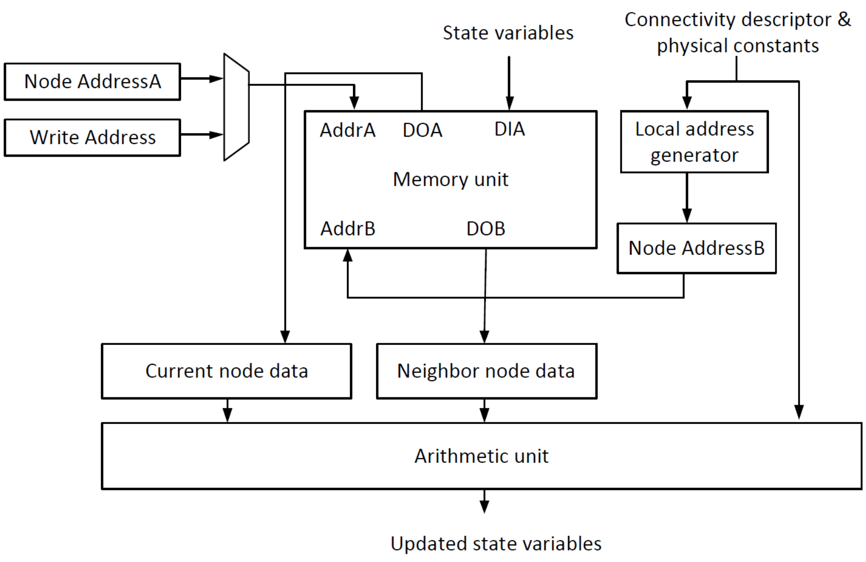

Figure 3.22. Block diagramm of the proposed dataflow processor unit ([J1]).

The main difference between structured and unstructured mesh computing is the additional knowledge of neighbors in the structured case. For unstructured meshes connectivity is not a trivial rule, it has to be stored and transferred to the processor unit. In Fig. (3.22) the dataflow processor unit of [J1] is shown with its input and output channels. During the explicit PDE computation, the corresponding state variables have to be updated at each mesh element at every timestep. Connectivity descriptors are also transferred through off-chip memory interface to a local address generator module which addresses the processor's Memory Unit which is a large FIFO that is filled continuously with mesh data.

Multiple dataflow processor units can be placed in a chain if there are enough resources on the FPGA chip. Fig. (3.23) presents the complete architecture with multiple pipelined processors on the same FPGA. The deeper levels need their memory units thus the increased number of these modules makes harder the limit on data locality. If data locality can be optimized better, it allows the usage of deeper pipelining on the same FPGA.

The arithmetic is pipelined according to the dataflow graph of the numerical algorithm. In the presented 2D cell-centered problem, each cell (triangle) has three interfaces, and the state variables are updated based on a flux function computed at the three interfaces. (For mathematical formulation, see [J1]) This arithmetic is optimized to reach the highest

DOI:10.15774/PPKE.ITK.2016.007

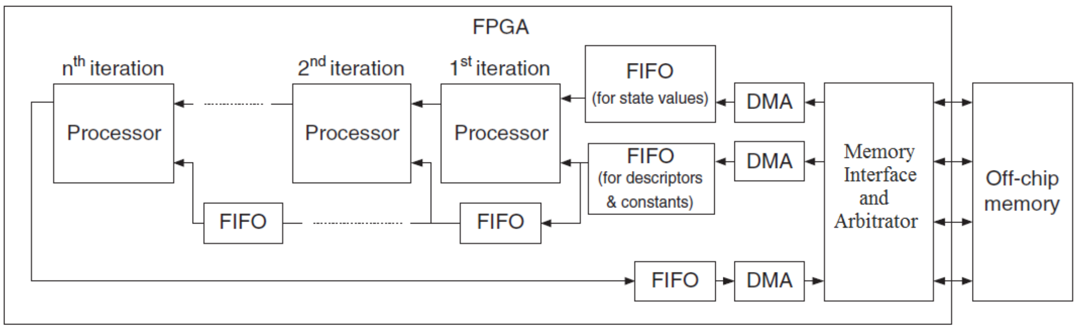

Figure 3.23. Outline of the proposed architecture. The processors are connected to each other in a chain to provide linear speedup without increasing memory bandwidth requirements. The number of processors is only limited by the available resources of the given FPGA ([J1]).

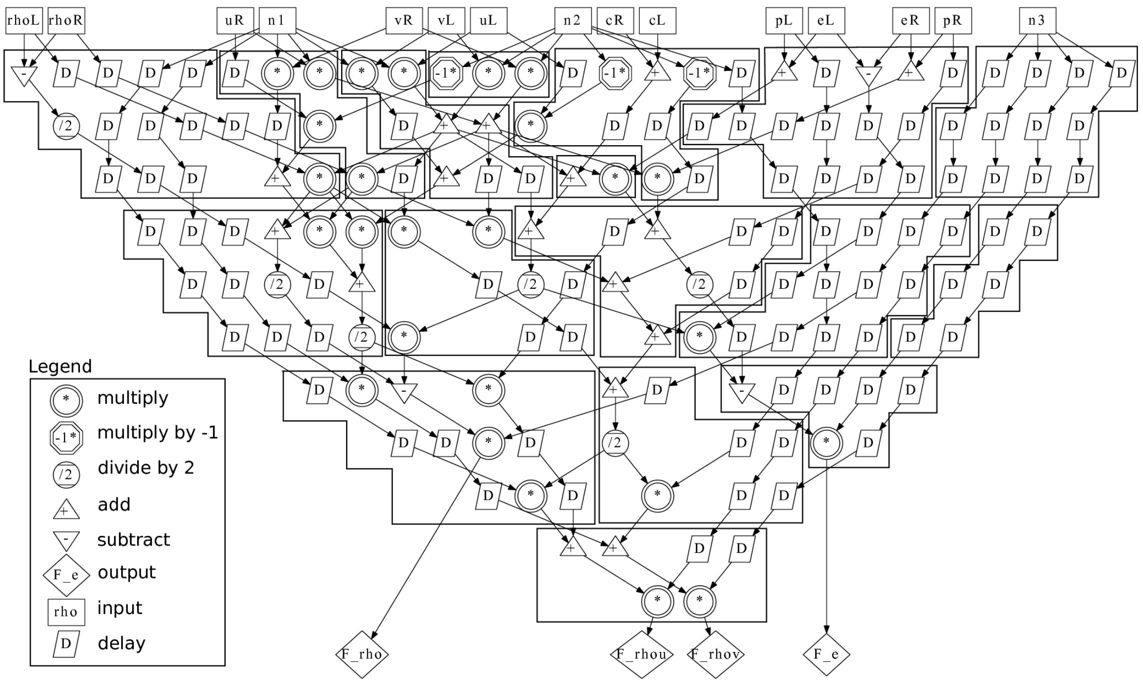

Figure 3.24. A partitioned data-flow graph generated from an explicit PDE solver numerical method and partitioned with the algorithm described in [R24]. Each part has its own local control ([C1]).

possible operating frequency. The dataflow structure is partitioned as can be seen in Fig. (3.24) where each part has its local control.

DOI:10.15774/PPKE.ITK.2016.007

3.4.3. Implementation of the DM for unstructured meshes

In the project of [J1] I was responsible for the memory bandwidth optimization part, which was performed by a special mesh node reordering and virtual partitioning method. In this section, I present our implementation results on an AlphaData ADM-XRC-6T1 reconfigurable development system equipped with a Xilinx Virtex-6 XC6VSX475T FPGA and 2 Gbyte on-board DRAM. This FPGA architecture was introduced in 2010, thus its performance is compared to the CPU and GPU architectures from 2010. I also investigate the effect of increased data locality on CPU and GPU performance.

The architecture was implemented using Xilinx and AlphaData IP cores at double precision. The optimized arithmetic unit for dissipation-free, inviscid, compressible fluid dynamics computation (cell-centered 2D) had 325 MHz maximum clock frequency. The AU performs a cell update in 3 clock cycles. Computation of each new state value requires loading and storing of one state variable vector (2x32 byte), loading of the area of the triangle (8 byte), and loading of three connectivity descriptors (3x26 byte) that are 150 byte altogether. Therefore, a 16.3 Gbyte/s memory bandwidth is required to feed the processor with valid data in every third clock cycle. However, our four 32-bit wide memory banks running on 800 MHz providing 12.8 Gbyte/s peak theoretical bandwidth. This limitation can be removed by slightly modifying the architecture shown in Figure (3.23) and connecting two Memory Units to one AU creating two virtual processors. One Memory unit is enabled in even clock cycles, whereas the other is enabled in odd clock cycles. In this case, one physical AU computes 2 time iterations, thus the necessary input bandwidth decreases to 8.2 Gbyte/s. This technique requires more on-chip memory resources for each physical AU, thus the data locality requirement of the dataflow architecture increases. It means that the mesh elements have to be reordered to provide lower graph bandwidth.

Table (3.1) summarizes the resource needs of the architecture that is shown in Fig. (3.22).

Table 3.1. Area requirements of the architecture.

	DSP	LUT	FF
Number of elements	525	43754	61936
XC6VSX475T utilization	26%	14.7%	10.4%

The most limiting factor is the number of DSP slices which enables 3 physical AUs on the Virtex-6 XC6VSX475T FPGA. In case of three processors, maximum bandwidth of

DOI:10.15774/PPKE.ITK.2016.007

the adjacency matrix of the mesh is 14,848 nodes; however, to avoid memory bandwidth bottleneck on our prototyping board, three AUs and six memory units have been implemented reducing the maximum bandwidth to 6,144 nodes.

Three clock cycles are required to update the state of one triangle; therefore, the performance of one processor is 108.3m triangle update/s. Computation of one triangle requires 213 floating-point operations; therefore, the performance of our architecture is 23.08 GFLOPs. On the Virtex-6 XC6VSX475T FPGA, three AUs can be implemented and connected in a pipeline resulting in 69.22 GFLOPs cumulative computing performance.

The performance of our architecture was compared with both a high-performance Intel

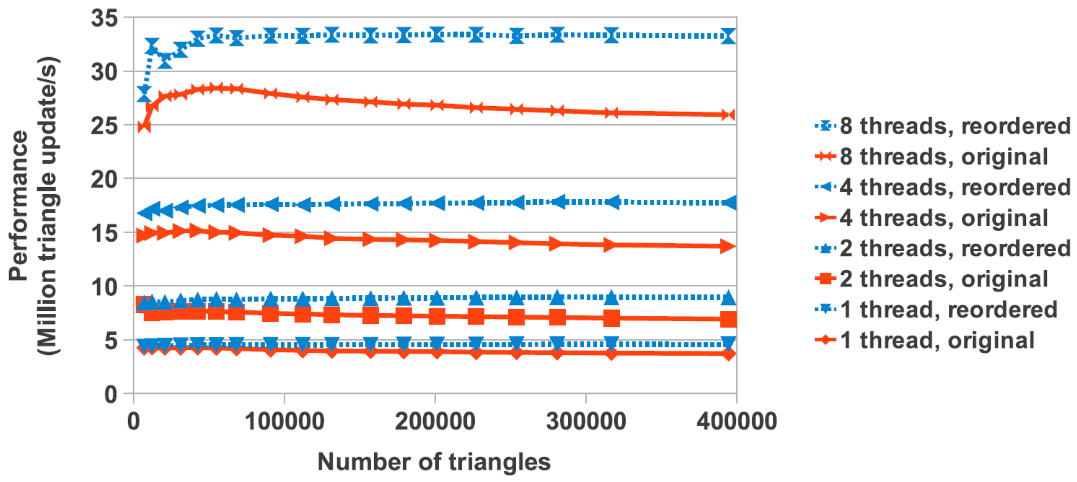

Figure 3.25. Measured performance of Intel Xeon E5620 microprocessor by using 1, 2, 4, and 8 threads ([J1]).

Xeon E5620 microprocessor and an Nvidia GeForce GTX 570 graphics card. The performance of the Intel Xeon E5620 microprocessor by using 1, 2, 4, and 8 threads is shown in Figure (3.25). As we expected, without reordering, the performance is decreasing as the mesh size is increased, whereas reordering preserves the performance. In case of the largest mesh and 8 threads, the reordered case outperforms the original one by 28.2% and reaches 33.22m triangle update/s or equivalently 6.86 GFLOPs.

The average performance of the simulator over various mesh sizes is shown in Figure (3.26) to investigate the speedup caused by the increasing number of threads. The average performance scales well with the number of threads in both the reordered and the original case; however, in both cases, the speedup compared with a single thread remains below

DOI:10.15774/PPKE.ITK.2016.007

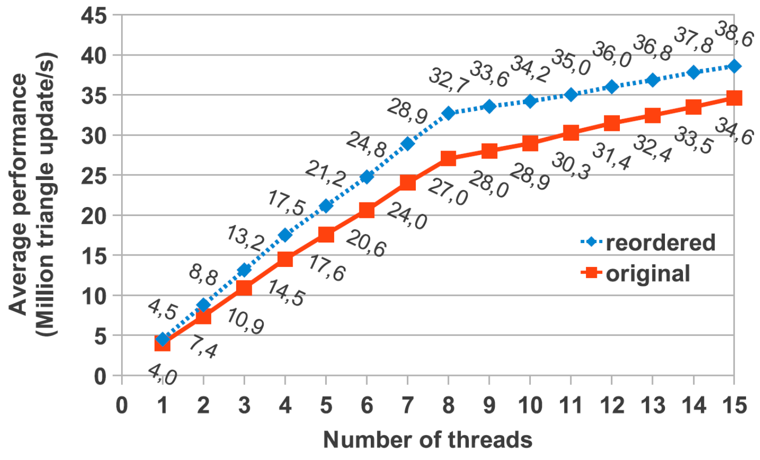

Figure 3.26. Measured average performance of Intel Xeon E5620 microprocessor over various mesh sizes using different number of threads ([J1]).

the number of threads. The peak performance of the CPU with 16 threads on reordered data is 38.6m triangle update/s which is nearly 9 times less than the 3 × 108.3m triangle update/s performance of our architecture.

Measurements were also performed on an Nvidia GeForce GTX 570 graphics card, which

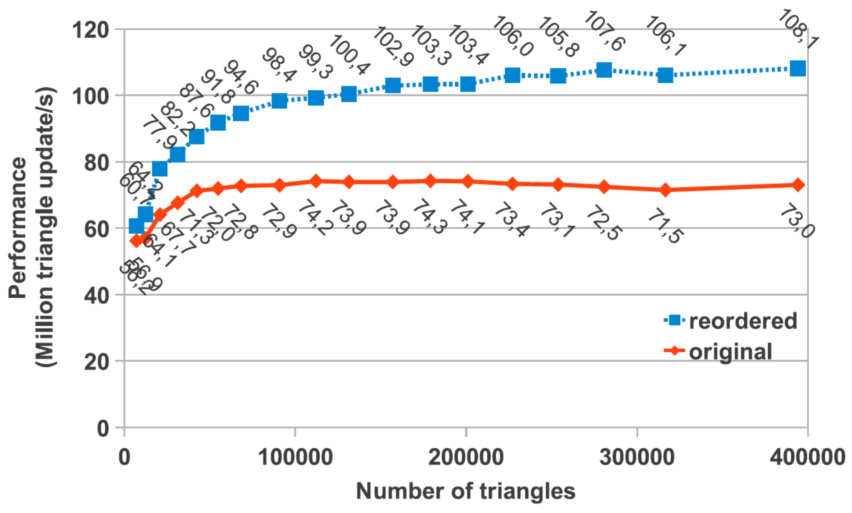

Figure 3.27. Measured performance of NVidia GTX570 GPU ([J1]).

has 480 cores running on 1464 MHz frequency, and 1280 MB GDDR5 memory with 152 GB/s bandwidth. The GPU program consists of a simple framework and a kernel, which computes a full triangle update. Figure (3.27) indicates that data locality improvement has large impact on the GPU performance. In the case of the largest mesh,

DOI:10.15774/PPKE.ITK.2016.007

the application with reordered input outperforms the original input by 48% and reaches 108.12m triangle update/s or equivalently 23.02 GFLOPs. The GPU was still 3 times slower than our deeply pipelined dataflow architecture.

Data locality improvement gives speedup for the CPU and the GPU but in the case of our dataflow architecture, it has a strict bound. If the input data stream fulfills the data locality constraint, the presented architecture outperforms both CPU and GPU in the case of 2D cell-centered fluid dynamics simulations.

In the paper [J1] only 1 FPGA chip took part in the computation. In the case of multiple FPGA-s, a special partitioning method is required for unstructured meshes which can handle the data locality bound inside the submeshes. As I presented in the previous chapter, the data locality and interprocessor communication have to be considered together. In the next chapter, I introduce the existing graph partitioning techniques and some findings of the relation of data locality and interprocessor communication in mesh partitioning.

DOI:10.15774/PPKE.ITK.2016.007

4. Chapter

Static Mapping

Mapping computation to k physical processors is a well-studied problem, with several solution techniques. Mapping can be defined as labeling of a process graph, where processes of the program are vertices and edges represent data dependencies between processes. Processes with identical labels are placed on the same physical processor. If a mapping does not change during the computation, it is called static mapping.
The goal of mapping is to reach maximum speedup with k processors against a single processor solution. This optimization problem is NP-complete in general because it can be traced back to the Multiprocessor Scheduling Problem which is known to be NP-complete [R5]. Many suboptimal heuristic solutions were developed for specific architecture topologies, such as hypercube [R25, R26]. In the case of having a complete graph as communication network topology, the static mapping is equivalent to the graph partitioning problem [R27].

4.1. Graph Partitioning

The k-way partitioning problem is the following: Given a graph $G(V, E)$, with vertex set V ($|V| = n$) and edge set E. A partition $Q = \{P_1, P_2, .., P_k\}$ is required, such that $\bigcup_{i=1}^{k} P_i = V$, $P_i \bigcap P_j = 0$ for $i \neq j$, the subsets have balanced size $|P_i| \approx n/k$, and the total number of edges between vertices belonging to different P_i subsets (edge cut) is minimal.
Size balance of P_i subsets provides balanced workload for all processors, and the minimized edge cut minimizes the communication between processors. This objective function

DOI:10.15774/PPKE.ITK.2016.007

may lead to a poor real speedup because the topology of processors is not taken into consideration [R26, R28].

Best known solvers are based on the Multi-Level (ML) scheme which is a general partitioning routine. ML has a partitioner as an input parameter and has three phases: coarsening, partitioning and uncoarsening. The size of input graph is decreased to a relatively small graph in coarsening phase. Coarsening is performed by node and edge unification steps. The input partitioner creates a partition of the coarsest graph in partitioning phase. In uncoarsening phase, the created partition is projected back to the original graph.

K-way partitioning is often solved by recursive bisection. Bisecting divides the input graph into two equal sized subsets, with minimized edge cut. With $log_2 k$ steps k subsets are created. This technique is very simple, and has some limits on solution quality [R29], however, it has been found effective in practice. There are few alternative methods, which perform the k-way partition directly. One class of them uses the ML paradigm to get the k-way partition [R30], with contracting until the number of vertices in the coarsest graph is the same as the number of subdomains.

4.1.1. Bipartitioning methods

Bipartitioning methods play an important role in k-way partitioners, a brief overview on them is necessary before generalized graph partitioning problems are introduced.

Spectral: The Laplacian matrix of graph G is L=D-A, where D is the degree matrix and A is the adjacency matrix of G. The elements of L are $l_{ii} = deg(v_i)$ and $l_{ij} = -1$ if $(v_i, v_j) \in E$ $l_{ij} = 0$ otherwise. L is positive semidefinite and has rank n-1 if G is connected. The eigenvalues of L are $0 = \lambda_1 < \lambda_2 \leq ... \leq \lambda_n$. An n element vector x is a separator if $x_i = \{-1, 1\}$ and $\sum_i x_i = 0$. It can be shown that $x^T Lx$ is 4 times the edge cut of separator x. The eigenvector, which corresponds to the second eigenvalue is v_2, which minimizes the edge cut. A separator vector x is chosen which is closest to v_2. The corresponding computational problem has high complexity thus many attempts were made to improve the speed of spectral methods [R31, R32, R33].

DOI:10.15774/PPKE.ITK.2016.007

Spatial Coordinate: If spatial coordinates are available, bisection can be performed based on the coordinates of vertices. An axis is chosen, which has the widest coordinate range. The vertices are sorted according to this axis, and the first half of them is assigned to the first subdomain, and the rest to the second. This method assumes that the grid resolution is nearly the same for all axis.

Inertial: A variant of the coordinate method. Each node has unit mass; the axis is chosen for which the moment of inertia of the nodes is minimized.

Greedy Graph Growing: A starting node is given, which forms a subdomain. A node is added to that subdomain iteratively, which leads to the minimal increase in the communication cost function. If balance reached the process stops, and another starting node is chosen. The output is the best separator.

Gibbs-Poole-Stockmeyer: GPS method was created for sparse matrix reordering. In the first stage, two endpoints of a pseudo-diameter of G is calculated. Breadth-first search is started from one of the endpoints until size balance reached.

Band: This is a refinement strategy; it needs an initial partition. Form the separator, a breadth first search is used to define a band graph. The separator is optimized inside this band. The banded graph is much smaller than the input problem, thus expensive heuristics can be used to create better solutions.

Exactifier: If load balance of a solution has to be improved, exactifier method can be applied. It moves nodes to the smaller subdomain in a greedy fashion.

Fiduccia-Mattheyses: Almost-linear variant of Kernighan-Lin iterative improvement method [R34]. It performs node swaps which decrease the cost function of the partition until a local optimum is reached.

4.1.2. Generalizations of Graph Partitioning

Graph partitioning in its original form does not handle many important factors. Generalized partitioning models have been created to satisfy these needs. All generalizations use additional information about the processor architectures and the corresponding communication topology.

DOI:10.15774/PPKE.ITK.2016.007

4.1.2.1. Hybrid Architecture:

If the processor nodes have different computational capabilities, the workload has to be distributed according to processing powers, thus $|P_i| = pow_i \cdot n$, where pow_i is the normalized computational capability of processor i.

4.1.2.2. Heterogeneous Processes:

Processes in V can have different computational complexities, described by a weight function $w_v(v_i)$. In this case, the workload of a processor is the sum of weights inside the corresponding subdomain. Real communication needs can be modeled by a weight function on the edges $w_e(e_{ij})$.

4.1.2.3. Multi-Constraint Partitioning:

Multiple balancing constraints can be modeled by using weight vectors instead of simple weights. For instance, weight can be defined as the computation need and another for the memory need [R35].

4.1.2.4. Skewed Partitioning Model:

The model can be improved by adding some penalty functions (skew) to the cost function. Let $p(v_i)$ be the part P_l in $Q = \{P_1, P_2, .., P_k\}$ to which v_i is assigned, and $d_{P_j}(v_i)$ is the desire of the vertex v_i to be in P_j. The cost function Eq. (4.1) should be minimized.

$$\sum_{e_{ij}} \begin{cases} w_e(e_{ij}) & \text{if } p(v_i) \neq p(v_j) \\ 0 & \text{otherwise} \end{cases} - \sum_{v_i} d_{p(v_i)}(v_i) \tag{4.1}$$

Desire functions can be used to hold additional knowledge about good solutions [R36].

4.1.2.5. Target Graph Representation:

Target or architecture graph representation gives an opportunity to model real communication costs [R37]. A target graph T is given, which has physical processors as its vertices $V(T)$ and real communication links as its edge set $E(T)$. Both process graph G and target graph T have weight functions defined on their vertices $Gw_v(v_k)$, $Tw_v(v_l)$ and edges $Gw_e(e_{ij})$, $Tw_e(e_{kl})$. Two functions are required: $\tau_{G,T} : V(G) \rightarrow V(T)$ and

DOI:10.15774/PPKE.ITK.2016.007

$\rho_{G,T} : E(G) \to \mathcal{P}(E(T))$, where $\mathcal{P}(E(T))$ denotes the set of all simple loopless paths which can be built from $E(T)$. Data exchanges between not adjacent processors, require transmissions through a route $\rho_{G,T}(e_{ij})$, which results additional cost. In communication cost function Eq. (4.2) every communication weight is multiplied by the length of its route.

$$f_C(\tau_{G,T}, \rho_{G,T}) = \sum_{e_{ij} \in E(G)} Gw_e(e_{ij}) |\rho_{G,T}(e_{ij})| \tag{4.2}$$

4.2. Sparse Matrix Reordering

The adjacency matrix A of a graph G is defined as $a_{ij} = 1$ if $e_{ij} \in E(G)$ $a_{ij} = 0$ otherwise. Matrix reordering transforms A by the product $A' = PAP^T$, where P is a permutation matrix.

In many applications, large $Ax = b$ linear systems have great importance. Efficient solvers use the Cholesky factorization $A = L^T L$ where L is a lower triangular matrix. The goal of many matrix reordering methods is to minimize the number of nonzero elements in the Cholesky factor L. Cholesky factorization exists only for positive definite A matrices, otherwise, $A = LU$ factorization is used where U is an upper triangular matrix. The second class of reordering methods transforms A to a narrow banded matrix, which increases data locality.

Mapping assigns a physical processor to each process, but the schedule of processes which belong to the same processor is still not defined. The local memory placement of data structures is also not defined. With reordering techniques, the schedule of processes and the memory placement of data structures can be determined. These questions become more and more important because novel processor architectures are hardly limited by memory bandwidth. Many improved partitioning models were developed, but none of them can handle data locality directly. A model is needed, which can consider memory bandwidth limitation and inter-processor communication together.

4.3. Data locality and interprocessor communication

The relation of mesh structure and communication need has been studied since the beginning of the development of graph partitioners. The communication need is proportional to the edge-cut of a submesh, which is geometrically the surface of the given submesh. It is

DOI:10.15774/PPKE.ITK.2016.007

easy to observe, that an abstract sphere has the best surface to volume ratio. This section gives a brief mathematical investigation of data locality and mesh structure dependencies. Data locality of a mesh is defined by the corresponding graph bandwidth.

4.3.1. Description of Graph Bandwidth Minimization and related work

Let $G(V, E)$ be a graph with vertex set $V(|V| = n)$ and edge set E. Labeling is a function f which exclusively assigns an integer from interval $[1, n]$ to each vertex, i.e., $f(v) = f(u)$ if and only if $u = v$ where $u, v \in V$. Let $N(v)$ denote the set of vertices which are adjacent to v.

1. Definition. *The bandwidth $B_f(v)$ of a vertex v respect to labeling f is*

$$B_f(v) = \max_{u \in N(v)} |f(v) - f(u)|. \tag{4.3}$$

2. Definition. *The bandwidth of a graph G respect to labeling f is the maximum of vertex bandwidths:*

$$B_f(G) = \max_{v \in V} B_f(v) \tag{4.4}$$

3. Definition. *The bandwidth of a grapf G is*

$$B(G) = \min_f B_f(G). \tag{4.5}$$

Minimal bandwidth labeling of a graph was shown to be NP-complete by Papadimitriou [R6], so exact methods can not be applied to large problems. Many heuristic algorithms were developed from the well-known Cuthill-McKee (CM) algorithm [R10] to recent metaheuristic approaches for instance a simulated annealing [R38], or a Tabu search [R39]. One of the most promising metaheuristics regarding solution quality is the GRASP (Greedy Randomized Adaptive Search Procedure) with Path Relinking [R40]. The most practical solution is the GPS (Gibbs, Poole, and Stockmeyer) method [R11], which is one of the fastest heuristics with good solution quality. Metaheuristic methods give better solutions, but their runtime is many times higher than runtime of the GPS. The GPS algorithm was given in 1976, afterward, there were many attempts to improve the original method, see Luo et al. [R41]. Most of the improved GPS heuristics have higher time complexity, which is an important issue in the case of large meshes which arise in practice.

DOI:10.15774/PPKE.ITK.2016.007

Fast reordering methods make mesh adaptation possible (refinement where it is necessary), but in this work, I assume a static mesh. Speed of the reordering method remains important because it is used for graph bandwidth measurement (good upper bound) in my proposed bandwidth limited partitioners.

4.3.2. Connection between Graph Bandwidth and Mesh Structure

The exact value of graph bandwidth $B(G)$ can not be efficiently calculated, but lower and upper bounds can be given for investigating dependencies between $B(G)$ and the mesh structure.
Any labeling can be used to get an upper bound on $B(G)$. A lower bound can be obtained as follows.

4. Definition (Graph diameter). *For every pair of vertices u, v let $d_{min}(u, v)$ denotes the length of the shortest path between u and v. The diameter of a graph G is*

$$diam(G) = \max_{\{u,v\} \in V^2(G)} d_{min}(u, v) \tag{4.6}$$

1. Theorem. $B(G) \geq \left\lceil \frac{n-1}{diam(G)} \right\rceil$

Proof:

Let f be an optimal labeling of G. The total change of labels along a path from arbitrary u to v is $|f(u) - f(v)|$. Therefore, there are two consecutive vertices where the labels change by at least $\left\lceil \frac{|f(u)-f(v)|}{d(u,v)} \right\rceil$. Let u and v be the vertices for which $f(u) = 1$ and $f(v) = n$. The compulsory change in the corresponding shortest path is $\left\lceil \frac{n-1}{d_{min}(u,v)} \right\rceil$. The highest possible value of $d_{min}(u, v)$ is $diam(G)$, thus in the shortest path from $f(u) = 1$ to $f(v) = n$ there must be two consecutive vertices, which labels differ in at least $\left\lceil \frac{n-1}{diam(G)} \right\rceil$.
■

A structured mesh of a rectangle ($a \leq b$) and labeling f are given on Figure 4.1. $B_f(G)$ of this mesh with labeling f is 11 ($a + 1$ in general), because this is the maximum difference between labels, which correspond to adjacent nodes. $diam(G)$ is 17 ($b - 1$ in general). According to Theorem 1 $B(G) \geq \lceil \frac{179}{17} \rceil = 11$, thus labeling f is optimal.
The bound of Theorem 1 is weak in some cases, but can be efficiently computed for every graph. If the above mesh contains only vertical and horizontal edges, a weaker lower bound

DOI:10.15774/PPKE.ITK.2016.007

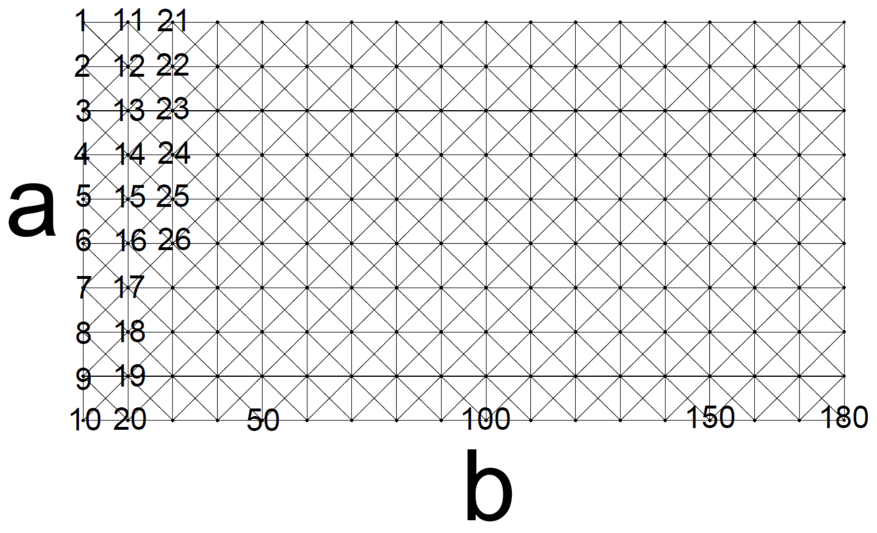

Figure 4.1. Example of structured mesh labeling. a=10, b=18, $B_f(G) = 11$, the minimal size of on-chip cache is BW=23. Node 14 can be updated, if [3..25] elements are in the on-chip memory.

can be obtained, because $diam(G)$ changes to $a - 1 + b - 1 = 26$. In the following theorem we show that this labeling f is still optimal if $a \leq b$.

2. Theorem. *Given an $a \times b$ rectangular grid G, i. e., $a \cdot b$ vertices with horizontal and vertical connections. If $a \leq b$ then $B(G) \geq a$.*

Proof: According to an arbitrary f we can start to index the vertices from 1 in increasing order. Let i be the first index, when all vertices in a column or in a row become indexed. First assume that a row is fulfilled. Then there is no fulfilled column, thus in every column must be an indexed vertex which is adjacent to an unindexed one. There are b different columns, thus the smallest index of these indexed vertices can not be grater than $i - b + 1$. The label of an unindexed vertex according to f can not be smaller than $i + 1$, thus the difference between them must be grater or equal to b.
Assuming that first a column is fulfilled, using the same argument we get that the minimum difference is a.
Since $a \leq b$, the statement of the theorem is proved. ■

Based on these simple examples, we can observe that $B(G)$ can be independent of the size of the problem. If columns are added to the example mesh on Fig. 4.1, $B(G)$ remains the same. In geometric view, assuming structured grids, ideal mesh shapes have a lengthwise direction like a long $a \times a \times b$ rod ($b \gg a$), which can contain a large

DOI:10.15774/PPKE.ITK.2016.007

number of elements with low $B(G)$. Worst shapes are sphere and cube because these shapes have no lengthwise direction, which can consume nodes with no extra bandwidth need. Unfortunately, these shapes have the lowest surface to volume ratio, which is proportional to the communication need.

For a partitioner both data locality and inter-processor communication are important. Unfortunately, there is a conflict between them because the minimization of inter-processor communication leads to abstract spheres, which are bad for data locality minimization. Tools are needed, which can provide tradeoffs between the two optimization goals.

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

5. Chapter

Bandwidth-Limited Partitioning

5.1. Problem definition

Based on the experiences with Dataflow Machines, I introduce Bandwidth-Limited Partitioning. The main goal of partitioning methods is to give a distribution of computation and data among physical processor nodes, which leads to minimal computation time. The goal of BLP is slightly different. BLP aims for high processor efficiency, which means in some case longer computational time because fewer processors with higher efficiency can be slower than much more processors with lower utilization.

BLP has four inputs: a mesh $G(V, E)$, a bound on communication to computation ratio $COMM\_Bound$, a bound on data locality BW_Bound, and the number of available processors K. The two bounds are determined by the parameters of the processor architecture as described in Section 2.3.

In BLP, k the number of utilized processors is also optimized, because a k-way partition is required which fits to the constraints. k is restricted by the COMM_Bound constraint and K the number of available processors.

5. Definition (Bandwidth-Limited Partitioning). *Given a graph $G(V, E)$, with vertex set V ($|V| = n$) and edge set E. BW_Bound, $COMM\_Bound$ and K are given parameters. A partition $Q = \{P_1, P_2, .., P_k\}$ is needed which maximizes the number of parts k considering the following conditions:*

Let $Out(P_i)$ denotes the set of outgoing edges of P_i.

$$k \leq K \tag{5.1}$$

DOI:10.15774/PPKE.ITK.2016.007

$$\max_i \left\{ \frac{|Out(P_i)|}{|P_i|} \right\} \leq COMM\_Bound \tag{5.2}$$

$$\max_i \{2 \cdot B_{f_i}(P_i) + 1\} \leq BW\_Bound \tag{5.3}$$

$$|P_i| \approx \frac{n}{k} \quad \forall\, i \tag{5.4}$$

Bounds on inter-processor communication (5.2) and data locality (5.3) provide the desired efficiency. Size balance is described by equation (5.4).
In equation (5.3) I assume the simplest discretization stencil with one explicit iteration, which means $s = 3$ and $Iterations = 1$ in Eq. (2.2). In case of other values of $Iterations$ and s, BW_Bound in Eq. (5.3) has to be modified according to Eq. (2.2). Because of the constrained nature of BLP, it is possible that there is no solution. In the case when BLP has no solution, one of the bounds must be relaxed to a higher value.
BLP claims partitions with optimized communication to computation ratio and data locality.

5.2. AM1 partitioning method

This section introduces a special and fast reordering method which aims at ordering f with minimized $B_f(G)$ while a proper estimation of bandwidth need is provided. Based on the estimation feature of the reordering method it can be used as a bandwidth limited partitioner. In this section, the notations of Sec. (4.3) are used in definitions and equations. Matrix bandwidth minimization and graph bandwidth minimization have the same meaning because of the strong connection between graphs and their adjacency matrices.

5.2.1. AM1 reordering method

The connections of data locality and graph bandwidth and the existing graph bandwidth minimization techniques are shown in Sec. (4.3). Here I present a simple constructive reordering method which has proper estimation of graph bandwidth during its labeling procedure.

DOI:10.15774/PPKE.ITK.2016.007

5.2.1.1. Two data locality bounds based on graph bandwidth

Using the definition of graph bandwidth, the size of the on-chip memory can be given as $(B_f(G) \cdot 2 + 1) * sizeof(data\ element)$. $(B_f(G) \cdot 2 + 1)$ is called central bandwidth (C_BW) because it assumes that the central element of the on-chip memory buffer has to be updated and the stream is moving continuously. If the input stream can be stopped and optional element of it can be updated we get another data locality descriptor which is called serial bandwidth (S_BW).

6. Definition (Serial-Bandwidth). *Serial bandwidth of a graph can be given as the maximum distance of nonzeros in a row of its adjacency matrix. With basic notations:*

$$s(i) = MIN\{f(v) : v \in N(u), f(u) = i\}$$

$$e(i) = MAX\{f(v) : v \in N(u), f(u) = i\}$$

$$S(i) = MIN\{s(i), s(i+1), ..., s(n)\}$$

$$E(i) = MAX\{e(1), e(2), ..., e(i)\}$$

$$S\_BW = MAX_i\{E(i) - S(i)\}$$

Based on the adjacency matrix it can be seen that $S\_BW \leq 2 \cdot B_f(G)$ and $B_f(G) \leq S\_BW$. With the definition of $C\_BW = 2 \cdot B_f(G) + 1$ we get the relation of these two data locality descriptors:

$$S\_BW \leq C\_BW - 1 \leq 2 \cdot S\_BW \tag{5.5}$$

5.2.1.2. Algorithm for bandwidth reduction

Several methods have been shown in the literature for minimizing $B_f(G)$. In this section, I define Amoeba1 (AM1) algorithm for direct serial bandwidth minimization. Our goal is to create a fast, effective constructive method which has proper, easy to calculate S_BW bounds in each construction step(details in next subsection). The method can be easily modified to handle C_BW based optimization.

Notations and definitions AM1 is a constructive method, in which a solution element is chosen and labeled in each step. Solution elements are the vertices of the input mesh,

DOI:10.15774/PPKE.ITK.2016.007

and the method grows a part till all of the vertices are covered.

Figure 5.5. shows the structure of a solution part P with n elements. Each node(i) has

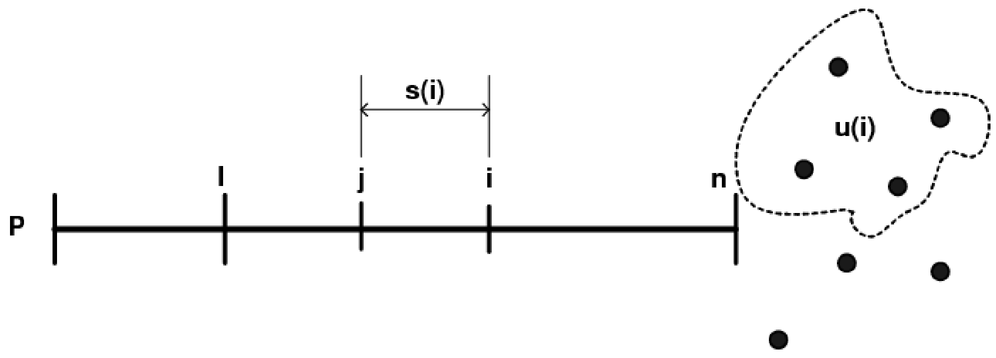

Figure 5.1. Structure of solution part P.

three base parameters: local_index=i, s(i), u(i).

s(i): is the distance between node(i) and its lowest indexed neighbor in the part: $s(i) = MAX\{i - j : j \in N(i)\}$.

u(i): is the set of nodes which uncovered by P, but must be added in later steps because of node(i): $u(i) = \{v : v \in N(i)\ AND\ v \notin P\}$.

I: is the index of the first elemet which has not empty u() set, so for every node(i) where $i < I$ all neighbors covered by P.

With these parameters, we can give bounds on the serial bandwidth. In AM1 method I use a simple lower bound for describing the importance of node(i):

$$imp(i) = (n - i) + |u(i)| + s(i)$$

This is obviously a lower bound on serial bandwidth need of the current node, because if we add node $v \notin u(i)$ to part P we still have to add all elements of u(i) to the part. For every node(i) $i < I$ imp(i)=0, because these nodes have all of their neighbors involved, so their effect on bandwidth does not depend on the later decisions.

5.2.1.3. Description of AM1

AM1 algorithm has two base steps: finding a starting vertex, and the labeling loop. The result is an ordering of the vertices.

DOI:10.15774/PPKE.ITK.2016.007

Finding a starting vertex The quality of the result of constructive bandwidth-reduction heuristics depends on which is the starting vertex. In GPS method, the authors presented a simple and effective solution for this problem. They gave an algorithm which returns the two endpoints of a pseudo-diameter. The AM1 algorithm uses this subroutine for finding the starting vertex.

Choosing a solution element AM1 Alg. (1) selects a node from u(I), which has a neighbor in P with maximal importance. Because all nodes in u(I) has node(I) as its neighbor, only $l \neq node(I)$ neighbors take part in the search. AM1 adds the candidate to

Algorithm 1 AM1 - Choosing a solution element

```
1: $candidate \leftarrow random\ element\ of\ u(I)$
2: $global\_max \leftarrow 0$
3: for $\forall k \in u(I)$ do
4:     for $\forall l \in N(k) :\ l \in P\ and\ l \neq node(I)$ do
5:         if $l.imp() > local\_max$ then
6:             $local\_max \leftarrow l.imp()$
7:     if $local\_max > global\_max$ then
8:         $candidate \leftarrow k$
9:         $global\_max \leftarrow local\_max$
10: return $candidate$
```

the part with index=n+1, and chooses the next element till the whole mesh is indexed. AM1 performs a kind of breadth-first indexing.

5.2.1.4. Results and conclusions

AM1 is a simple constructive algorithm for large problems, I compare its results to the fast and effective GPS method. As mentioned earlier, better quality algorithms exist for bandwidth reduction, but these methods can not be applied to large meshes($\geq$ 100.000 vertex) because of their complexity. Test cases are generated by Gmsh with different mesh density parameters which appear in the names of the example meshes. The cases showed on Table 5.1 comes from 2-dimmensional meshes, with assigning a vertex to each triangle and an edge between vertices which are represent adjacent triangles, so we get a mesh with maximal_degree = 3. These meshes appears when we use finite volume solver during the solution of a partial differential equation. In these low-degree cases AM1 provides similar solution quality (S_BW) to GPS, in 4% less time. Serial bandwidth need of the reordered

DOI:10.15774/PPKE.ITK.2016.007

mesh data defines the necessary on-chip memory requirement which has to be fulfilled in the case of a dataflow machine. The running time of both methods depends on the number of vertices, and the structure of the mesh(finding a starting vertex). The results

Table 5.1. Results of Amoeba1 method compared to GPS.

Case	N	S_BW GPS	S_BW AM1	GPS time(s)	AM1 time(s)
step_2d_bc_cl30	7063	122	122	0,078	0,052
step_2d_bc_cl40	12297	176	175	0,154	0,109
step_2d_bc_cl50	20807	253	227	0,175	0,1
step_2d_bc_cl70	42449	359	341	0,633	0,49
step_2d_bc_cl90	68271	481	506	0,998	0,785
step_2d_bc_cl110	112093	569	591	2,144	1,955
step_2d_bc_cl130	157099	740	738	1,59	1,316
step_2d_bc_cl150	201069	794	805	3,239	3,094
step_2d_bc_cl170	252869	972	923	4,316	3,92
step_2d_bc_cl190	316715	1030	1082	5,913	5,707
step_2d_bc_cl200	394277	1093	1155	5,855	5,532
step_2d_bc_cl320	930071	1923	1809	17,035	18,687

S_BW: serial-bandwidth of solutions, N: number of vertices.
Algorithms tested on one core of an Intel P8400 processor

for high-degree(20-30) cases can be found on Table 5.2. These cases generated from the same complex 3D geometry, by increasing the density of the mesh. I found that GPS is 29% superior on these general instances, but 13% slower than AM1. These results show that the difference is not increasing with the complexity of the problems. AM1 is proposed for bandwidth limited access pattern generation in the case of dataflow machines. The goal of this comparison is to demonstrate that the main advantages of the leading reordering method (GPS) are preserved in AM1. AM1 is proposed for large problems where the reordered mesh data still have too high S_BW need. For the architecture in [J1] the S_BW limit is 6144 which can not be satisfied for the three largest problems on Table 5.2. AM1 can handle these cases too with a simple extension which does not affect the solution quality of the method.

5.2.2. AM1 as a partitioner

In the case of large problems it is possible, that the renumbered mesh has larger bandwidth than the available on-chip memory, these cases should be handled too. Here I show an AM1 based method, which generates an input order which has at most a pre-specified serial-bandwidth. In the input order every vertex is executed once, but can be loaded many times,

DOI:10.15774/PPKE.ITK.2016.007

Table 5.2. Results for 3D high-degree cases

Case	N	S_BW GPS	S_BW AM1	GPS time(s)	AM1 time(s)
3d_075	3652	381	391	0,279	0,27
3d_065	5185	500	763	0,144	0,107
3d_055	8668	712	764	0,655	0,587
3d_045	15861	1066	1478	0,468	0,42
3d_035	33730	1880	1863	2,209	2,22
3d_025	88307	3384	3443	7,569	6,42
3d_018	244756	6582	10110	39,509	27,598
3d_015	417573	9066	14930	85,797	59,958
3d_012	519983	20561	23554	413,72	383,075

so we need a flag **Ex** to store it is only a ghost node(false) or have to be calculated(true). Serial bandwidth in this job means the following: for every Ex=true vertex, all of their neighbors surely be in the on-chip memory when the execution reaches them. If the defined bound is less than the serial-bandwidth for whole mesh provided by the AM1 method, the input will be r-times longer, where $1 \leq r$. The bound on bandwidth obviously has to be more than the maximal_degree of the graph.

5.2.2.1. AM1 based bounded S_BW method

The main concept of handling the bounded bandwidth is the usage of a proper serial bandwidth estimation, which is available in AM1 method. When a Part reaches the S_BW bound, the process calculates which vertices can be executed, and calls the AM1 method for the rest of vertices where Ex is not true. The main process starts new AM1 parts until all vertices are Ex=true. The output of the method is an access pattern(list of {index,Ex} pairs), where all vertex has true execute flag once, and can be appear many times as a ghost node.

Estimation of serial-bandwith: Given an AM1 Part, the task is to estimate its serial bandwidth. AM1 estimates the part's bandwidth in each construction step. If $i < I$ for node(i) in part P, it has all of its neighbors inside the part, so if the bandwidth is less than the bound when I becomes larger than k, node(k) can not increases S_BW anymore. As shown earlier imp(i) is a lower bound on serial-bandwidth, but in the estimation a proper upper bound is required for the stopping condition. Because in all step AM1 adds a node from u(I) to the part, we can calculate more than a proper upper bound for a node(i), we

DOI:10.15774/PPKE.ITK.2016.007

can give the exact value.

$$S\_BW(i) = (n-i) + |\bigcup_{I \le k \le i} u(k)| + s(i) \quad I \le i \tag{5.6}$$

Eq. 5.6 is equivalent to the definition of serial bandwidth[1] Equation 5.6 could be a stopping condition, but the proposed method has a less complex and useful upper bound for stopping decision defined in Eq. 5.7.

$$S\_BWBound \ge \underbrace{MAX}_{I \le k \le n} imp(k) \tag{5.7}$$

If Eq. 5.7 holds, AM1 continues to add nodes to part P, stops otherwise. This condition is not an upper bound for the whole part, but it provides in every step that node(I) has lower serial-bandwidth than the given bound. If I jumps to a higher index I' after AM1 adds node(n) to P, we can be sure that for all node(i) $I \le i \le I'$ serial bandwidth is under the bound, because $\bigcup_{I \le k \le I'} u(k) = \{node(n)\}$, so $imp(i) = S\_BW(i)$ inside the range [I ,I'].

Finalizing a Part: When Eq. 5.7 not holds, the proposed algorithm finalizes the part and starts a new instance of AM1 on the rest of the not executed nodes. Finalization has two tasks: it has to label vertices which are executed in part P, and have to label vertices which has only executed neighbours because these nodes can be cut out of the mesh(I call them perfect nodes Pr=true). Ex=true and Pr=false vertices have to be loaded again because they have at least one Ex=false neighbor. In AM1 imp(i)=0 and u(i)={} for all node(i) for which Ex=true.

$$s^* = \underbrace{MIN}_{I \le k \le n}\{k - s(k)\}$$

Algorithm 2 AM1 - Finalizing a Part

```
1: for $\forall k \in P$ do
2:     if $k.local\_index < s^* \quad and \quad k.Ex \mathrel{!=} true$ then
3:         $k.Pr \leftarrow true$
4:     if $k.local\_index < I$ then
5:         $k.Ex \leftarrow true$
```

[1]the authors have a proof

DOI:10.15774/PPKE.ITK.2016.007

5.2.2.2. Results and conclusions

It is obvious that the proposed algorithm generates access patterns which have lower S_BW than a given bound. The input length multiplier **r** is a good parameter for measuring the solution quality. (r-1)*100% of the vertices have to be reloaded from the main memory, but the processing still has 0% cache-miss[1].

Measurements on three meshes with different S_BW bounds can be found on Table 5.3.

Table 5.3. Results of AM1 bounded bandwidth optimization

Case	AM1_BW	S_BW Bound	num. of parts	N	overall length	r	time(s)
3d_075	391	392	1	3562	3562	1	0,255
3d_075	391	380	4	3562	4288	1,203	0,392
3d_075	391	300	9	3562	4945	1,388	0,7
3d_075	391	200	20	3562	5929	1,664	1,148
3d_035	1863	1864	1	33730	33730	1	2,317
3d_035	1863	1800	6	33730	38053	1,128	7,78
3d_035	1863	1500	5	33730	38702	1,147	4,439
3d_035	1863	500	88	33730	58171	1,724	36,095
3d_015	14930	14931	1	417573	417573	1	70,108
3d_015	14930	14000	2	417573	431081	1,032	77,004
3d_015	14930	10000	2	417573	427211	1,023	71,278
3d_015	14930	7500	8	417573	449441	1,076	91,058
3d_015	14930	5000	34	417573	476170	1,140	53,247
3d_015	14930	2500	130	417573	557190	1,334	687,385

AM1_BW: the bandwidth provided by AM1 for the whole mesh
overall length: length of the generated access pattern
N: number of vertices, Algorithm tested on one core of an Intel P8400 processor

The results show that the solution quality (r-factor) mainly depends on the distance of the S_BW bound from the S_BW need of the mesh and also depends on the maximal_degree of the mesh. This is a really good news because maximal_degree is around 20-30 for a typical 3D mesh, while the S_BW bound is around 10-40k[2] nowadays and increasing with each new generation of FPGA-s. The number of generated parts determine the time consumption of the proposed method because in each restart of AM1 the algorithm calculates the pseudo-diameter for the rest of the graph. The results on 3d_015 show that we can go below 25% of the original S_BW with 15-30% reload. This method gives the opportunity of deciding the size of the on-chip memory synthesized to the FPGA, so the designers can have more free area with sacrificing computational time.

[1]assuming the handling of multiplicity problems
[2]the bound is depending on sizeof(data element)

DOI:10.15774/PPKE.ITK.2016.007

AM1 is beneficial for accelerators with 1 FPGA chip because the size of possible input graphs becomes much higher. However, in high performance computing the usage of multiple accelerator chips is a must. If AM1 is used as a partitioner for multi-chip solutions the interprocessor communication and size-balance constraints can not be controlled.

5.3. Depth-Level Structure based partitioning

If the reordered input has greater on-chip memory requirement than the available resources, or we want to use more FPGA chips, the input mesh must be divided into parts.
Famous partitioning methods, for instance, METIS[R42], minimize the edge-cut between the parts and balance the size of the generated parts. The size-balance is important because each part is given to a multi-processor, and the overall runtime is determined by the multi-processor which gets the largest part. The edge-cut is proportional to the communication required between the processors. Graph bandwidth of the resulting parts is smaller than the graph bandwidth of the whole mesh, but the methods do not deal with the graph bandwidth directly.
The graph bandwidth of the resulting parts is important because it determines the minimal size of on-chip memory, which is necessary for maximal data reuse. The edge-cut is also relevant because it is proportional to the number of random accesses, which appear when the PE reads data from adjacent parts (ghost nodes).
In many cases, the covering surface (set of boundary nodes) of the mesh is also known, which gives information about the geometry, but not used by traditional partitioners. In this section a novel partitioning method is shown, which creates parts with minimized graph bandwidth, using geometrical information derived from the cover. The proposed method is an example, which presents new possibilities in mesh partitioning.

5.3.1. Depth Level Structure (DLS) Based Bisection

DLS is a hidden structure in every unstructured mesh for which the covering node set is defined. Depth is the distance from the cover. Nodes of the mesh with same depth belong to a level. Nodes in the deepest levels represent the critical areas of the mesh in case of bandwidth minimization.

DOI:10.15774/PPKE.ITK.2016.007

5.3.1.1. Objective

The primary goal of DLS-Based Bisection is to reduce the bandwidth of the resulting parts. However, the objective of bandwidth minimization alone is meaningless because the bandwidth of the resulting parts can be decreased optionally by increasing the edge-cut. In Figure 5.2 a 2D example is shown, where the bandwidth of the parts are reduced with large edge-cut. The edge-cut is also important because communication between the processors is proportional to the edge-cut.
Reducing the bandwidth of the parts with acceptable communication requirement is the objective of the DLS-Based method. The acceptable communication requirement (edge-cut) is an application-specific parameter. In novel FPGA array, the cost of reading data from the off-chip memory of an adjacent FPGA is usually 10 times slower than reading from its off-chip memory. In the case of using Alpha-Data ADM-XRC-6T1 cards, the theoretical memory bandwidth is 12.8 Gbyte/s inside, and 1.25 Gbyte/s between the cards[R43][1]. When only 10% of the whole memory accesses are external reads and their occurrences are balanced, the whole memory bandwidth can be utilized to feed PEs.

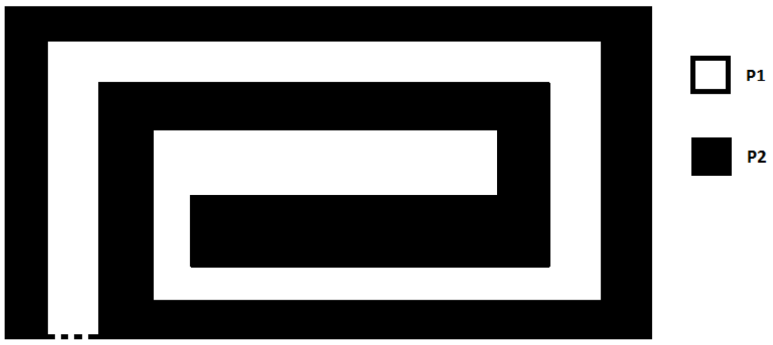

Figure 5.2. 2D example where good bandwidth reduction partitioning leads to unacceptable communication need.

5.3.1.2. Basic Entities and Operations

DLS-Based partitioning uses some subroutines which are general tools for manipulating node sets. The most important node set is the covering surface, furthermore, the separators are also node sets in our nomenclature. These operations are based on waves (breadth-first search - BFS) which are starting from a set of nodes and spreading through the

[1]the number of adjacent parts is limited

DOI:10.15774/PPKE.ITK.2016.007

mesh. Spatial waves of BFS are useful to get node sets, which can be used as surfaces or separators.

Cover Set of nodes belonging to the covering surface.

Deepest(DLS) Set of nodes in the deepest levels of DLS. The Deepest set contains the three deepest levels of the DLS structure. An example is shown in Figure 5.3.

Level_Structure(in: in_set, out: LS) Generates a level-structure from *in_set*. LS is a series of sets (levels), where the elements of *in_set* form the zero level, and the rest nodes associated to the level according to their minimal distance from *in_set*.

Level(in: node, LS) A function which returns the level index of node in LS.

Pseudo_Diameter(in: in_set, out: (u,v)) Gives the two endpoints of a pseudo-diameter on *in_set*. The method is similar to the first step of GPS method, returns two points which have maximal distance from each other.

Grow(in: start_node, border_set, out: out_set) Grows a set from *start_node*, by adding the neighbors of included elements into the set. An element is added if it has no node from *border_set* as its neighbor. Grow is a kind of diletation, for which *border_set* is a bound.

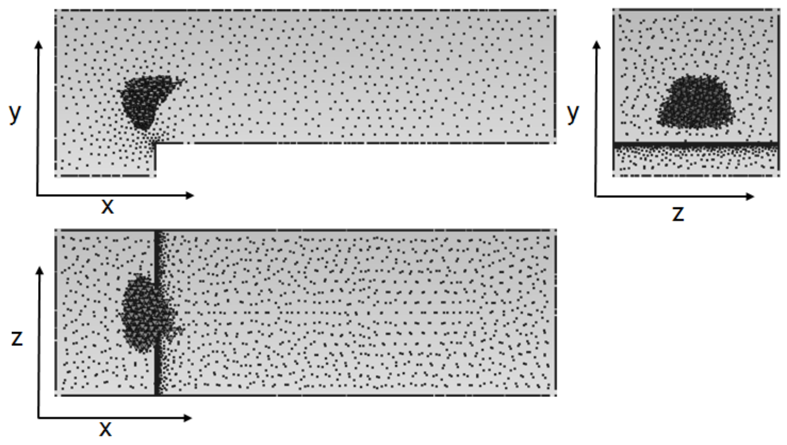

Figure 5.3. Deepest set of tunnel202, xyz projections are shown. Points are the vertices of covering surface, nodes of Deepest set represented by tetrahedrons.

DOI:10.15774/PPKE.ITK.2016.007

5.3.1.3. DLS Bisection

The base concept of DLS bisection is the division of the mesh along the deepest set of nodes. Here I show the method that is presented in [C2]. GPS method creates level-structures from a boundary node and indexes nodes level by level. Bandwidth of the solution is proportional to the size of the largest level. In geometrical view, the ordering starts from a boundary surface and creates onion skins through the mesh, and the bandwidth is proportional to the largest cutting surface. In the case of a structured grid of a rectangle, the bandwidth of GPS solution is proportional to the smaller side, which is often optimal. The elements of Deepest set take place on a line which is perpendicular to the smaller side. Furthermore, the line separates the rectangle into two equal sized parts.
DLS can be obtained by the Level_Structure() routine, starting the BFS from the *Cover* set, the resulting level structure will be the DLS structure. The Deepest set is the union of the three deepest levels in DLS. The method uses the three deepest levels, because the deepest level may contain only one node, and the base idea of the bisection is to cut the deepest area of the mesh. Using pseudo-diameter routine, the method gets two endpoints of *Deepest* set, which have maximal distance from each other in the whole mesh(*Deepest* set is not necessarily connected). The DLS-Based bisecting method generates the separating surface in two steps. In the first stage, a set of nodes is obtained which have the same distance from the two endpoints of the *Deepest* set's pseudo-diameter (Alg. 3). The resulting set is used during the second stage. The final set separates the mesh into two parts.

The resulting separator is parallel to the pseudo-diameter of the *Deepest* set, but the

Algorithm 3 Get Separator

```
Precondition: Cover
1: $Level\_Structure(Cover, DLS)$
2: $Pseudo\_Diameter(Deepest(DLS), (u, v))$
3: $Level\_Structure(\{u\}, LU)$
4: $Level\_Structure(\{v\}, LV)$
5: $sep1 = \{x : Level(x, LU) - Level(x, LV) \in \{0, 1\}\}$
6: $Pseudo\_Diameter(sep1, (u, v))$
7: $Level\_Structure(\{u\}, LU)$
8: $Level\_Structure(\{v\}, LV)$
9: $sep2 = \{x : Level(x, LU) - Level(x, LV) \in \{0, 1\}\}$
10: return $sep2$
```

DOI:10.15774/PPKE.ITK.2016.007

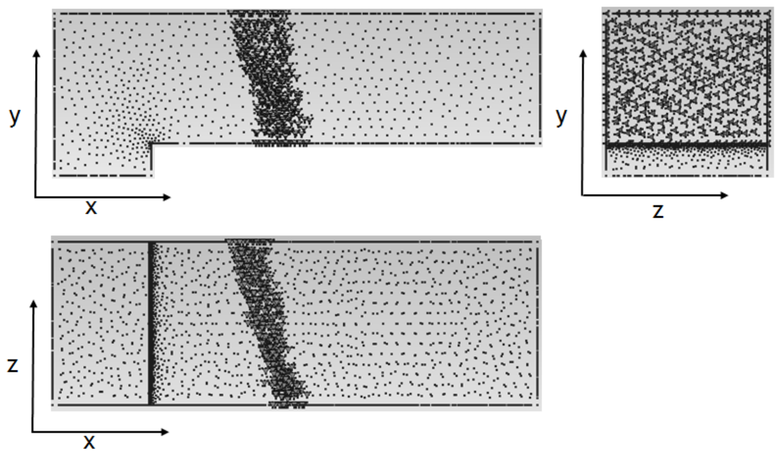

Figure 5.4. sep2 of tunnel202, xyz projections are shown. Points are the vertices of covering surface, nodes of the separator represented by tetrahedrons.

separator not necessarily intersects the *Deepest* set. The correction step (Alg. 4) is responsible for placing the separator to the middle of the *Deepest* set. Separator *sep2* is the set of nodes which have the same distance from the endpoints of the pseudo-diameter of *sep1*. If the *Deepest* set is closer to one of the two endpoints, the separator must be moved towards that point. All levels of the level-structure started from *sep2*, is equal to two separators which appear at the sides of *sep2*. The correction phase removes the further node set from each level. The level is chosen as corrected separator which has the largest intersection with *Deepest* set. The partition method is not completed by determining the separator

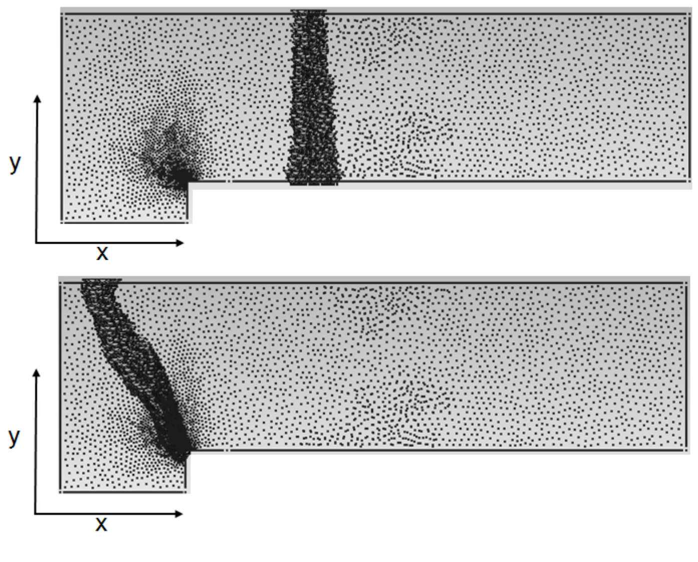

Figure 5.5. Separator of tunnel100 before correction (up), and after (down).

surface, because our separator is a set of nodes, therefore the partition is ambiguous. Two

DOI:10.15774/PPKE.ITK.2016.007

Algorithm 4 Separator Correction

Precondition: Separator **sep2**, nodes **u** and **v** with corresponding level sructures **LU** and **LV**

```
 1: $dist\_u \leftarrow MIN\{Level(x, LU) \mid x \in Deepest\}$
 2: $dist\_v \leftarrow MIN\{Level(x, LV) \mid x \in Deepest\}$
 3: if $dist\_u < dist\_v$ then
 4:     $Level\_Structure(\{u\}, LS1)$
 5:     $Level\_Structure(\{v\}, LS2)$
 6: else
 7:     $Level\_Structure(\{v\}, LS1)$
 8:     $Level\_Structure(\{u\}, LS2)$
 9: $Level\_Structure(sep2, LSep)$
10: for $\forall x \in LSep$ do                              ▷ Remove the further node set
11:     if $Level(x, LS1) > Level(x, LS2)$ then
12:         $delete\ x\ from\ LSep$
13: $Separator \leftarrow level\ L\ of\ LSep\ for\ which |L \cap Deepest|\ is\ maximal$
14: return $Separator$
```

parts are obtained by using the *Grow* subroutine, but the separator and its nearest neighborhood still remain unpartitioned. Unpartitioned nodes are added to the smaller part (Alg. 5). The method can be finished at this point, but the size balance is not guaranteed. A simple solution is to grow the smaller part till balance reached.

Algorithm 5 Get parts from separator

Precondition: Separator

```
1: $s \notin Separator$
2: $Grow(s, Separator, part1)$
3: $s \notin Separator \cup part1$
4: $Grow(s, Separator, part2)$
5: $rest = \{x : x \notin part1 \cup part2\}$
6: $Add\ rest\ to\ the\ smaller\ part$
7: $Grow\ smaller\ part\ till\ balance\ reached$
8: return $part1, part2$
```

5.3.2. Results

My test environment is based on Gmsh[R44], which is a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. In Gmsh simple 3D models can be defined, meshed and partitioned. The covering surfaces are given as physical surfaces, which makes possible for our algorithm to get the cover of the mesh.

DOI:10.15774/PPKE.ITK.2016.007

The test models are shown in Figure 5.6. Meshes generated from *sgrid* are structured grids of hexahedrons, from *snake* and *weight* unstructured tetrahedron-based meshes are generated with uniform density, in case of *tunnel* the density of the mesh is increased around the step. A comparison is shown in Table 5.4 between the DLS-Based and the METIS-

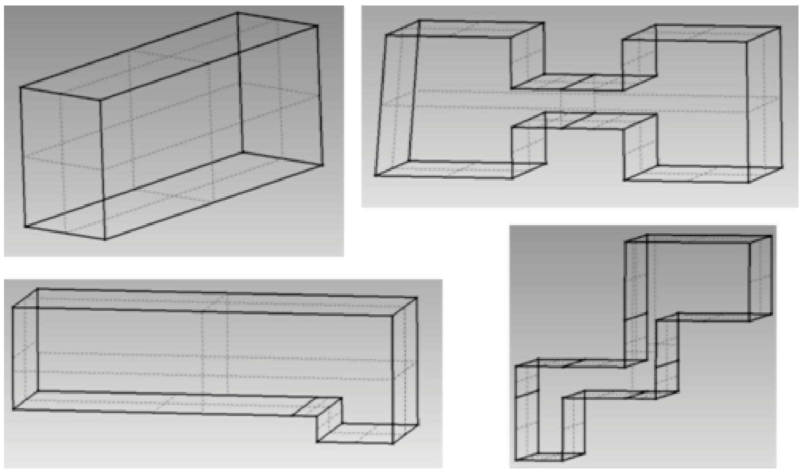
Figure 5.6. Test shapes. sgrid, weight (up), tunnel, snake(down)

recursive partitioning. The objectives of the two partitioners are different, but there is no other known method, which minimizes the graph bandwidth of the resulting partitions, and METIS is one of the most popular solvers. METIS solutions are generated by the sequential implementation because its solution quality (edge-cut or external reads) is better or equal to the quality of parallel partitioning schemes for instance ParMETIS [R45]. Novel results of mtMETIS aim for better running times on parallel architectures and better scaling [R46, R47] but their solution quality has not been improved.

For the structured brick-shaped problems, the DLS method provides in average 28% better BW partition, with acceptable communication ratio (external reads / inner). The COMM ratio is better for larger problems, in the case of *sgrid4*, which has 1M vertices, the COMM ratio is only 1,6%. The values depend on the ratio between the side lengths, but the tendencies are independent.

The communication ratio (edge-cut ratio) is getting better when the mesh density is increased for all problem instances. This is obvious because the cutting surface has N-1 dimension in the case of an N-dimensional mesh. This feature is important because DLS computes a kind of N-1 dimensional surface, which separates the mesh into two parts. DLS-Based solutions can have unacceptable communication need for small meshes, for

DOI:10.15774/PPKE.ITK.2016.007

Table 5.4. Results of DLS-Based Partitioning

Problem	N	Orig BW	MET BW	MET COMM	DLS BW	DLS COMM
sgrid1	2200	221	181	0,0388	141	0,1216
sgrid2	16800	841	641	0,0192	479	0,0628
sgrid3	131200	3281	2517	0,0087	1719	0,0323
sgrid4	1036800	12961	9967	0,0045	6599	0,0164
snake100	7821	777	689	0,0254	531	0,079
snake038	158544	5701	5371	0,0074	4941	0,0095
tunnel202	18210	2353	1385	0,0292	1303	0,0262
tunnel100	191592	12525	5675	0,017	5949	0,027
weight045	4899	641	581	0,0169	311	0,1764
weight022	35922	2363	2131	0,0078	1411	0,0559
weight012	230891	8087	8785	0,0037	8785	0,0075

N: number of vertices. Orig BW: GPS BW for the whole mesh.
MET/DLS BW: BW of partitions.
MET/DLS COMM: number of outgoing edges / number of internal edges

example *weight045*, where the COMM ratio is 17,5%.

Using DLS-Based bisection, the resulting partitions have 40% reduced graph bandwidth compared to the whole mesh, and create 20% better solutions than METIS. METIS minimizes the edge-cut and provides size-balance, the DLS-Based solutions have same size balance quality, however, the edge-cut is several times higher. There is a tradeoff between graph bandwidth and communication need, and DLS creates partitions with higher COMM ratio to provide reduced graph bandwidth.

DLS reduces the bandwidth by 40-50% by separating the mesh along the Deepest set, in case of tunnel geometry the METIS solution has the same reduction ratio, for *tunnel202* the partition is nearly similar, and DLS has better COMM ratio, for *tunnel100* the METIS solution has better graph bandwidth.

The difference between the traditional methods and DLS-Based partitioning can be observed on Figure 5.7, where the separator of *weight022* is shown. DLS-Based bisection cuts the weight shape in the longitudinal direction, instead of choosing the small edge-cut between the two weights. This partition leads to 34% better graph bandwidth reduction with 5,6% COMM ratio.

If the DLS separator does not provide size balance, the proposed method grows the smaller part until the balance is reached. This strategy can increase the graph bandwidth of the solution. For *weight012*, the BW of the resulting parts are 6741 without size balance, and 8785 after because the balance leads to a quasi-similar partition to METIS. The graph

DOI:10.15774/PPKE.ITK.2016.007

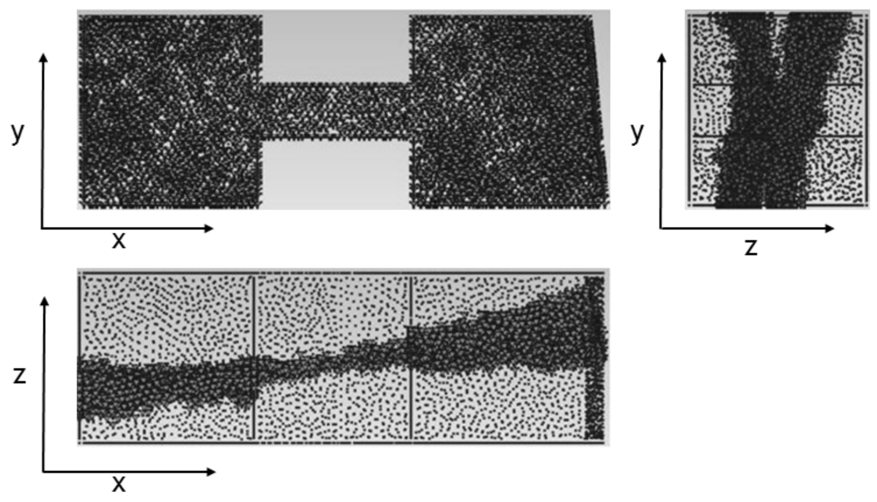

Figure 5.7. Separator of weight022.

bandwidth of the parts can be higher than the original mesh because GPS is a heuristic algorithm, and the graph bandwidth mainly depends on the largest cutting surface.

5.3.2.1. Bounded BW partitioning

The proposed method is created for providing partitions which meet constraints on the graph bandwidth. If the original mesh has larger GPS bandwidth than the given bound, the method bisects the mesh. Parts can be bisected if the constraint on the bound is still unsatisfied. The constraints on graph bandwidth can be reached in fewer bisection steps with the DLS-Based method, than with other traditional partitioners.

5.3.3. Conclusions

In this section, a novel partitioning approach was presented, which focuses on the bandwidth reduced node ordering of the resulting parts. The motivation of this kind of partitioning is based on FPGA designs, where the graph bandwidth of the input parts has to be below a bound. The proposed solver uses the covering node set of the given mesh, which contains information from the geometry without spatial coordinates. The covering surface is often given (physical simulations) but is not used by traditional partitioners. The proposed method is an example of using covering surface in partitioning techniques. The components of the DLS-Based method are operations on node-sets, which operations use spatial waves to compute abstract surfaces (node sets) inside the mesh.

DOI:10.15774/PPKE.ITK.2016.007

The results show that the proposed partitioning algorithm creates partitions with better graph-bandwidth quality than METIS, with acceptable edge-cut. The size-balancing steps of the method should be improved, because the graph-bandwidth quality of the solution can be damaged. The main shortcoming of this method is the uncontrolled edge-cut which has to be considered with data locality in BLP. The following sections give solvers for the BLP in the cases of structured and unstructured meshes.

5.4. BLP method for structured meshes

For special structured meshes near optimal BLP partitions can be obtained. Here we show optimal grid-type BLP partitions for $a \times b$ and $a \times b \times c$ structured grids, where only horizontal, vertical and depth connections are defined.

5.4.1. Grid-Type BLP Partitions

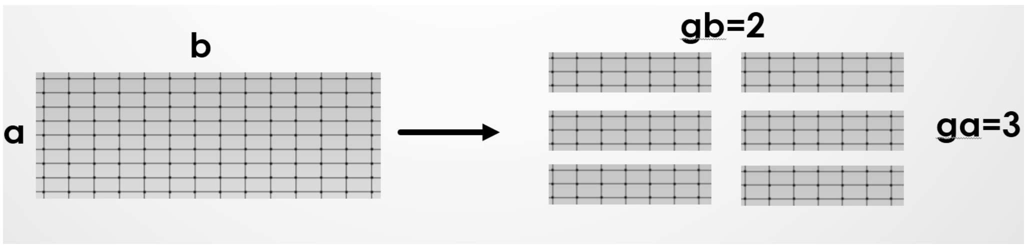

Figure 5.8. Example of Grid-Type partitoning. The structured grid is divided according to a $ga \times gb$ grid.

Grid-type partition of an $a \times b$ mesh is a $g_a \times g_b$ grid of uniform $\frac{a}{g_a} \times \frac{b}{g_b}$ submeshes, as it can be seen in Fig.(5.8). Let $s_a = \left\lfloor \frac{a}{g_a} \right\rfloor$, $s_b = \left\lfloor \frac{b}{g_b} \right\rfloor$ and $S_a = \left\lceil \frac{a}{g_a} \right\rceil$, $S_b = \left\lceil \frac{b}{g_b} \right\rceil$ denote the possible sizes of the sides of a submesh. A grid-type partition may contain submeshes of size $S_a \times S_b$, $S_a \times s_b$, $s_a \times S_b$ and $s_a \times s_b$. According to Theorem 2 the maximum graph bandwidth of these parts is $min\{S_a, S_b\}$.
$|Out(P_i)| = 2 \cdot s_a + 2 \cdot s_b$ if P_i is enclosed by other parts. If $g_a = 1$, $|Out(P_i)| = 2 \cdot s_a + 0 \cdot s_b$, because s_b sides are on the boundary of the mesh. If $g_a = 2$, $|Out(P_i)| = 2 \cdot s_a + 1 \cdot s_b$ because P_i has one s_b side which is adjacent to an other part. In $|Out(P_i)|$ only the non-boundary sides are accounted, each side is multiplied by $\{0, 1, 2\}$. Let m_a, m_b denote the maximum side multipliers. With these notations we get Eq. (5.8) as BLP goal function.

DOI:10.15774/PPKE.ITK.2016.007

In Eq. (5.9) I use s_a, s_b because this choice maximizes the left hand side.

$$\max_{\substack{k \leq K \\ (5.9)}} \{k = g_a \cdot g_b\} \tag{5.8}$$

$$\begin{aligned} 2 \cdot \min\{S_a, S_b\} + 1 &\leq BW\_Bound \\ \frac{m_a \cdot s_a + m_b \cdot s_b}{s_a \cdot s_b} &\leq COMM\_Bound \end{aligned} \tag{5.9}$$

The statement of Theorem 2 can be generalized to $a \times b \times c$ grids. In this case, graph bandwidth can not be smaller than the size of the smallest side ($S_a \cdot S_b$ or $S_a \cdot S_c$ or $S_b \cdot S_c$), and an ordering is also known, which can provide this graph bandwidth. Side multipliers m_{ab}, m_{ac} and m_{bc} should be defined. The BLP goal function in this case is Eq. (5.10).

$$\max_{\substack{k \leq K \\ (5.11)}} \{k = g_a \cdot g_b \cdot g_c\} \tag{5.10}$$

$$\begin{aligned} 2 \cdot \min\{S_a \cdot S_b, S_a \cdot S_c, S_b \cdot S_c\} + 1 &\leq BW\_Bound \\ \frac{m_{ab} \cdot s_a \cdot s_b + m_{ac} \cdot s_a \cdot s_c + m_{bc} \cdot s_b \cdot s_c}{s_a \cdot s_b \cdot s_c} &\leq COMM\_Bound \end{aligned} \tag{5.11}$$

Size of the solution space is K^2 for $a \times b$ and K^3 for $a \times b \times c$ grids. This search space can be efficiently reduced, if we consider that $g_a \cdot g_b \cdot gc \leq K$. In the case of an $a \times b$ grid we can start the search with $g_b = 1$ for which ga can take values from $\{1 \dots K\}$, then for $g_b = 2$ g_a can take values from $\{1 \dots \lfloor K/2 \rfloor\}$, and so on. After the $\lfloor K/2 \rfloor$-th step $g_a = 1$ remains the only one available value. The size of the reduced search space is (5.12).

$$K - \left\lfloor \frac{K}{2} \right\rfloor + \sum_{i=1}^{\lfloor K/2 \rfloor} \left\lfloor \frac{K}{i} \right\rfloor \tag{5.12}$$

Which is $\mathcal{O}(K \cdot ln(K))$ because of the behavior of harmonic series. This size is suitable for an exhaustive search, the optimal grid-type partition can be obtained. In case of $a \times b \times c$ grids, for each g_c value we have a subspace which has complexity of (5.12), where K is divided by g_c. This adds another harmonic series component, which leads to $\mathcal{O}(K \cdot (ln(K))^2)$ complexity, which is still suitable for exhaustive search.

DOI:10.15774/PPKE.ITK.2016.007

5.4.2. Quality of Grid-Type BLP partitions

In Section 4.3 I showed that partitions with best inter-processor communication have bad data locality (spheres vs. rods). Using Optimal Grid-Type Partitions, we can investigate this problem experimentally.

For the architecture presented in [J1] $COMM\_Bound = 0.1$ is a hard limit. If an $a \times a \times a$ cube is given, the smallest a which can satisfy this bound is $a = 60$ where $\frac{6 \cdot a^2}{a^3} \leq 0.1$, and this part has 216000 nodes. A part with 216K nodes is suitable for a kilo-processor dataflow unit, but the number of these units k is hardly limited by $COMM\_Bound$. For a 100M node $500 \times 500 \times 400$ mesh, the optimal Grid-type partition with $BW\_Bound = Inf$ $COMM\_Bound = 0.1$ $K = Inf$ is a $8 \times 8 \times 7$ grid partition, which has 448 dataflow processors and $BW = 7309$. The smallest part is a $62 \times 62 \times 57$ submesh which is almost a cube and has 219108 nodes. If $BW\_Bound = 5000$ is set, the result is a $12 \times 4 \times 7$ grid partition, which has only 336 processors. The smallest part is a $41 \times 125 \times 57$ submesh with 292125 nodes, and its shape has a lengthwise direction. For $BW\_Bound = 4000$ the partition contains $41 \times 45 \times 200$ rod-shaped submeshes. For $COMM\_Bound = 0.2$ the resulted partition can support 3468 dataflow units with $BW = 1801$.

METIS is one of the best partitioners for the minimization of inter-processor communication. The comparison of METIS and Optimal Grid-Type partitions shows the gap between METIS and the achievable partition quality. Measurements on small and medium sized structured grids are presented in Table 5.5. BW_Bound is set to $Infinity$ (Inf) or to the lowest possible value, which has Grid-Type solution with the given $COMM\_Bound$. In Grid-Type partitions, all bounds are satisfied. k_{opt} is the number of parts in the optimal grid-type partition, and MET_k is a k_{opt}-way METIS partition.

Results on the first three instances show the efficiency of METIS in the minimization of inter-processor communication. METIS solutions have only 0.004 difference from optimal grid-type communication ratios, and the graph bandwidth quality is also extremely good. This is possible because the minimal change in the $COMM\_Bound$ can result in significant change in the achievable graph bandwidth. With 0.002 relaxation of the METIS communication ratio, BW need can be reduced from 1377 to 799 for $37 \times 41 \times 77$ and from 2895 to 2133 for the $73 \times 81 \times 153$ grid. For larger instances like $100 \times 81 \times 200$ there is much more room for development. Optimal grid-type partitioner without BW_Bound reaches 0.059 better communication ratio (sphere) than METIS, better spheres have a

DOI:10.15774/PPKE.ITK.2016.007

Table 5.5. Comparison of METIS and Optimal Grid-Type Partitions

Instance	*BW*	*COMM*	Grid-Type	MET_k	MET_k
	Bound	*Bound*	k_{opt}	*BW*	*COMM*
19x21x39	Inf	0.1	2	621	0.05
19x21x39	419	0.1	2	621	0.05
37x41x77	Inf	0.1	4	1377	0.104
37x41x77	1483	0.1	4	1377	0.104
37x41x77	799	0.106	4	1377	0.104
73x81x153	Inf	0.1	12	2895	0.104
73x81x153	3035	0.1	12	2895	0.104
73x81x153	2133	0.106	12*	2895	0.104
100x81x200	Inf	0.1	20	3233	0.159
100x81x200	3281	0.1	20	3233	0.159
100x81x200	1641	0.159	20*	3233	0.159
100x81x200	1559	0.159	44	2039	0.222
100x81x200	799	0.222	44*	2039	0.222

*: K is fixed to the given value, otherwise K=Infinity (Inf)

slightly larger BW need. In the case of 20-way partitioning, with the same communication ratio, BW need can be decreased from 3233 to 1641, and from 2039 to 799 in the case of 44-way partitioning. These results show that direct data locality handling is important and possible. The available partitioning methods which decrease only the inter-processor communication create partitions which have bad shapes (sphere). Optimal grid-type BLP partitioning is a simple and efficient tool for partitioning the most important class of structured meshes.

5.5. BLP method for unstructured meshes

Bandwidth Limited Partitioning is a challenging task. My DLS-based algorithms which create partitions with good data locality often suffer from bad communication to computation ratios. The minimization of inter-processor communication is a well-studied field with effective existing solvers, for instance, METIS. Currently, the best way of handling BLP is a METIS-AM1 hybrid.

DOI:10.15774/PPKE.ITK.2016.007

5.5.1. METIS-AM1 hybrid method for handling BLP

METIS-AM1 has two stages. First the number of parts k is defined by multiple METIS trials. From $k = 2$ METIS recursive bisections are performed, until K is reached or the communication need exceeds $COMM\_Bound$. If the last partition exceeds $COMM\_Bound$, with interval halving method the largest k can be found, for which $COMM\_Bound$ is satisfied. At this point, a k-way METIS partition is given, which has better communication to computation ratio than the $COMM\_Bound$. In the second stage, METIS-AM1 calls AM1 to each METIS part.
Unfortunately, this method does not solve BLP but makes it possible to handle data locality and inter-processor communication together. $COMM\_Bound$ fully and BW_Bound virtually can be satisfied.

5.5.2. METIS-AM1 results on unstructured meshes

The main goal of Bandwidth-Limited Partitioning is to maximize k, the number of submeshes, for which the bounds on inter-processor communication and data locality are satisfied. k is mainly limited by the communication bound, thus it can be determined by METIS. If a k-way METIS partition has larger communication need than the communication bound, there is little hope to find a $(k + 1)$-way partition which fits to the conditions. METIS does not deal with data locality directly, but with sufficiently high k, the k-way METIS partition can fit to the BW_Bound.

For a dataflow machine, the input mesh must be partitioned according to the BW_Bound. In Table 5.6 a comparison between METIS-only and METIS-AM1 solutions for unstructured meshes is shown. Our test instances are generated by Gmsh [R44], which is an open source mesh generator. $COMM\_k$ is the largest k, for which the communication bound holds, and BW_k is the smallest k, for which the bound on data locality is satisfied in the k-way METIS partition (Fig. 5.9). For all instances, the first BW_Bound is set to the data locality need of the MET_{COMM_k} partition, thus no further partitioning is required. Then BW_Bound is decreased, which forces METIS-only to increase k, and METIS-AM1 to create virtual partitions.

The bound on the communication ratio allows only 2 or 4 parts for small and medium sized instances in Table 5.6. This does not generate any problem because each part is given to a kilo-processor dataflow unit. The results show that BW can be halved with AM1 at

DOI:10.15774/PPKE.ITK.2016.007

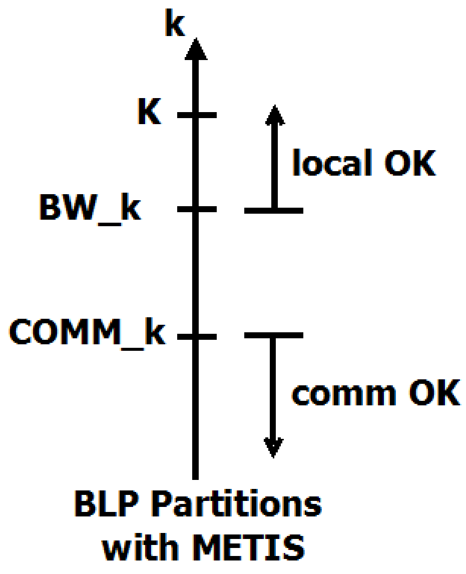

Figure 5.9. Relation of k-way METIS partitions to the bounds on data locality and inter-processor communication.

the cost of 10-20% data reload for larger meshes while METIS-only solutions have 0.2-0.3 communication to computation ratios. METIS-AM1 solutions have fewer dataflow units with high utilization and power efficiency while METIS-only solutions have more dataflow units, which suffer from inter-processor communication limits.

Typical values of BW_Bound is 30-100K for existing dataflow architectures. For these small and medium size problems, assuming $BW\_Bound = 30K$ $COMM\_Bound = 0.1$, METIS partitions which created in the first stage of METIS-AM1 are solutions for the BLP, and can be applied to dataflow units after GPS node reordering. However, parameters in Eq. (2.2) should be integrated into the BW_Bound. If the discretization stencil has $s = 5$ width, BW_Bound have to be divided by 2. On-chip memory resources of the FPGA are distributed among the on-chip cache, the pipelined arithmetic, and I/O buffers. In the case of a complex numerical method, BW_Bound should also be decreased to provide memory resources to other modules on the chip.

The BW_Bound becomes important, when K is relatively small: $\frac{n}{K} > 1M$. In these cases, BLP solutions are heavily limited by the BW_Bound, while $COMM\_Bound$ can be easily satisfied. Cases with relatively small K are important because dataflow units have 2-48 GB off-chip DRAM that enables them to handle large submeshes, which is needed to reach the best power efficiency [R13]. It may happen, that BLP has no solution for relatively small K-s. In these cases, METIS-AM1 can still create an applicable partition with node

DOI:10.15774/PPKE.ITK.2016.007

Table 5.6. Comparison of METIS-only and METIS-AM1 BLP Solutions

Instance	*n*	*BW Bound*	*COMM Bound*	*COMM k*	*BW k*	MET_{BW_k} *COMM*	*AM1 r − factor*
snake_02	33737	2329	0.1	2	2	0.067	1
snake_02	33737	2000	0.1	2	3	0.266	1.114
snake_02	33737	1200	0.1	2	8	0.634	1.370
snake_03	225041	6939	0.1	2	2	0.036	1
snake_03	225041	6000	0.1	2	3	0.158	1.124
snake_03	225041	3500	0.1	2	10	0.504	1.281
tunnel_01	79161	7641	0.1	1	1	0	1
tunnel_01	79161	7000	0.1	1	2	0.135	1.131
tunnel_01	79161	3700	0.1	1	3	0.222	1.147
tunnel_02	537690	19068	0.1	2	2	0.073	1
tunnel_02	537690	18000	0.1	2	3	0.124	1.063
tunnel_02	537690	9500	0.1	2	4	0.223	1.113
weight_04	171859	8046	0.1	2	2	0.026	1
weight_04	171859	7000	0.1	2	3	0.224	1.138
weight_04	171859	4000	0.1	2	6	0.298	1.209
weight_05	1243064	20811	0.1	4	4	0.099	1
weight_05	1243064	18000	0.1	4	5	0.132	1.056
weight_05	1243064	9000	0.1	4	11	0.246	1.0965
weight_05	1243064	9000	0.2	8	11	0.246	1.0961

orderings for the dataflow units.

5.6. Conclusions

In this chapter, the Bandwidth Limited Partitioning of meshes is presented. This is motivated by Dataflow Machines (DM) that can utilize total off-chip memory bandwidth with perfect caching. DMs need data streams with maximized data locality. BLP uses bounds on data locality which is connected to the available cache size. The ratio of inter-processor bandwidth and off-chip memory bandwidth is also used to define bound on communication to computation ratio. These architecture dependent bounds can ensure high processor utilization, which is the main goal of BLP. The optimization of the number of processors is an important novelty of BLP. The bound on communication ratio determines the maximum number of processors, which evades the waste of processor resources.

DOI:10.15774/PPKE.ITK.2016.007

With simple mathematical tools I showed that the goals of inter-processor communication and data locality are conflicting, thus existing communication minimization methods are not able to solve BLP effectively.

A modified reordering method, AM1 is presented for the generation of data locality bounded memory access patterns which are a kind of virtual partitioning (parts are given to the same physical processor). A new DLS-based mesh partitioning method is also presented which focuses on data locality property of the resulting parts, but can not deal directly with the interprocessor communication need.

Optimal Grid-type BLP partitioning is proposed for structured meshes, which time complexity $\mathcal{O}(K \cdot (ln(K))^2)$ is independent of the mesh size. Measurement results show that heavy data locality improvement can be achieved with minimal relaxation of the bound on inter-processor communication. METIS solutions can be surpassed in all parameters.

METIS-AM1 hybrid method is proposed for unstructured meshes. This method does not solve BLP, but creates partitions in which $COMM\_Bound$ fully and BW_Bound is virtually satisfied. If the given BLP has no solution, METIS-AM1 can still create an applicable partition with node orderings for the dataflow units.

Improved data locality is essential for DMs, and also important for other processor architectures. Inter-processor communication is still the most important factor in mesh partitioning, but there is great potential in considering data locality. In future work, I want to give complete solution for unstructured meshes, and consider the communication topology of the processors.

DOI:10.15774/PPKE.ITK.2016.007

6. Chapter

Applicable Partial Solution Generation for Fast-response Combinatorial Optimization

Combinatorial Optimization (CO) problems play an important role in many applications. Vehicle routing, production planning, resource assignment, or task scheduling problems require the best possible solution while the corresponding CO problems are $\mathcal{NP}$-hard. CO solutions contain decision variables for which optimized values are assigned. In many cases, it is beneficial to know the optimized values of a subset of these variables. For instance, if a robot has 100 elementary tasks and knows the first 5 tasks in the optimized schedule, it can start the execution. In a resource assignment case, the corresponding resources can be transferred to the assigned agent before the complete assignment is created.

Because of the $\mathcal{NP}$-hard nature of many CO problems, the most efficient solvers are heuristics which search for solutions in an exponentially growing solution space [R8]. The quality of solutions depends on the available time for the search, thus, the response time of the optimizer and solution quality are conflicting. Applicable partial solution generation provides a better trade-off between optimization time and quality because it makes possible to restrict response time only for a subset of decision variables instead of the termination of the whole optimization process when the required response time is reached.

In applicable partial solution generation (APSG), a fixed partial solution is required in constrained time, which is extendable to a feasible, complete solution. APSG has similari-

DOI:10.15774/PPKE.ITK.2016.007

ties to real-time online optimization, but there are differences. APSG differs from real-time optimization because it requires only a partial solution in constrained time. It is not similar to online optimization because the whole problem instance is known.
In APSG, the intermediate partial solution cannot be modified because it will be applied before the solver generates the complete solution. While APSG solutions reduce the response time of the optimizer significantly, fixed partial solutions restrict the solution space. Therefore, fast partial solution generation can harm the reachable optimality of the complete solution. If the goal function describes the execution time of operations, it is possible that the use of a partial solution leads to better overall time performance because running time of the optimizer is also accounted. Figure (6.1) shows an example, where execution time of the solution (optimality) is worse, but the corresponding overall time needed is better. Except these cases, fast-response APSG methods generate inferior complete solutions compared to slow-response high-quality methods.
Much effort has been made to define good constructive heuristics and high-quality comp-

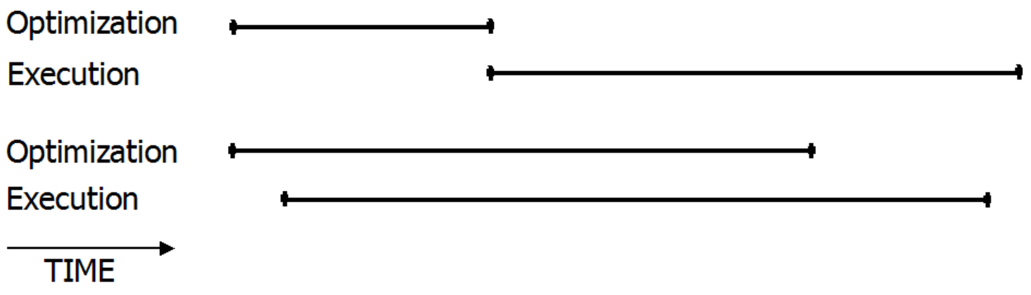

Figure 6.1. Task scheduling example: Execution started after a complete solution is generated (Up). Partial solution is applied before the optimization is completed. (Down)

lex approximation methods. These results should be used in APSG. Most of the novel approximation methods are defined as derivatives of problem-independent metaheuristics. Metaheuristic formulation makes the integration and cooperation of different methods easier [R7, R8]. A metaheuristic framework is needed, which can support fast algorithm component integration and highlights the central questions of APSG.
There are several definitions for metaheuristics. I prefer the following definition which has been given by M. Dorigo who is the author of Ant-Colony Optimization: ,,A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with rela-

DOI:10.15774/PPKE.ITK.2016.007

tively few modifications to make them adapted to a specific problem."
Basic metaheuristics are Local Search, Simulated Annealing (Kirkpatric 1983), TABU Search (Glover 1989), Variable Neighbourhood Search (VNS), Guided Local Search (GLS), Ant-Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Evolutional Optimization (Holland 1975, Fogel 1994), summary and references can be found in [R7].
In [C3] Variable Subset Merger (VSM) metaheuristic is proposed for APSG, which has subordinate heuristics to define the importance order of decision variables the desired value for a variable and the clustering of the variables.
This chapter investigates the possibilities and limitations of applicable partial solution generation with VSM.

6.1. Variable Subset Merger

A constrained optimization problem can be given by three components: a finite set X of decision variables X_i, $i = 1, \dots, n$, an objective function $f(X_1, \dots, X_n)$ and a set of constraints among the variables. These constraints can be precedence constraints given by a directed graph $P(X, R)$ for permutation-based problems, where edges in R describe precedence. System of equations and inequations $g_k(X) \leq c_k, k = 1, \dots, max_k$ and $h_l(X) = d_l, l = 1, \dots, max_l$ can also be given. Minimization or maximization of f over the decision variables is required considering the constraints.
Subsets of decision variables define subproblems. If some variables in a subset have assigned values, these variables define a subset solution. These solutions can be extended by assigning values to free variables in the corresponding subset. Subset solutions form together a partial solution to the original problem, which is applicable in the case when a complete, feasible solution exists that contains it. Variable Subset Merger (VSM) descri-

Algorithm 6 Variable Subset Merger

```
1: assign subset $S_i$ to every $X_i$
2: **while** number of subsets $\neq 1$ **do**                ▷ Main Loop
3:     all subsets recommend value for their unfixed variable
4:     select subset $S_i$
5:     fix recommended value for the unfixed $X_k \in S_i$
6:     select $S_j$ for $S_i$ , $j \neq i$
7:     $S_i = S_i \cup S_j$
8: set the last unfixed variable.
```

DOI:10.15774/PPKE.ITK.2016.007

bes the extension of multiple subset solutions until they form a complete solution. During the VSM process, each subset has one free decision variable and has fixed assignments for the rest of variables in it.
As a first step, the optimization problem is divided into n sub-problems; each contains only one decision variable. In all construction steps (Main loop) one decision variable is fixed. At the beginning all subsets recommend a value for their free decision variable (step 3), then a subset S_i is selected (step 4) and its recommendation is accepted (step 5). At this point it has no free variable, thus S_i should be merged with another subset S_j (step 6). After S_i and S_j were merged (step 7), the unified subset has exactly one free variable, and the number of subsets is decreased by one. After $n-1$ iterations, only one subset remains, which has $n-1$ fixed variable and one free variable.
I demonstrate the VSM method on the Travelling Salesman Problem (TSP), where the successor of a city is the decision variable. VSM has three subordinate heuristics. The first one is the subset solver heuristic (step 3), which defines the values for the recommendation. In our TSP example, the nearest neighbor heuristic can be used. The second heuristic handles the subset selection (step 4). The subset selection heuristic defines the order of importance of the free decision variables. The third heuristic selects a subset S_j, which is the most appropriate for S_i to merge with (step 6). Decision variables in TSP define S_j directly. The subset that contains the successor of S_i-s last city is the compulsory choice. Figure (6.2) shows a TSP example, where three subsets take part in the recommendation phase (left), and two remain after the construction step (right).

6.1.1. Variations of subset selection

When a recommendation is accepted, the corresponding subset solution generates additional constraints for the other subsets. In Fig. (6.2) for example, the extension of S_1 makes the recommendation of $X_8 = 1$ unavailable, because it would cause inner circle in the unified S_1. Subset selection defines which subset is the most important. The role of subset solving heuristics is to define the order of possible values for a free decision variable. Subset selection makes the connection between subsets. This heuristic handles the minimization of optimality loss that comes from fixed subsolutions. All subset selections have a connection to a branch and bound method because a fixed decision drops out a branch in the search tree. Next I list the basic variants of subset selection.

DOI:10.15774/PPKE.ITK.2016.007

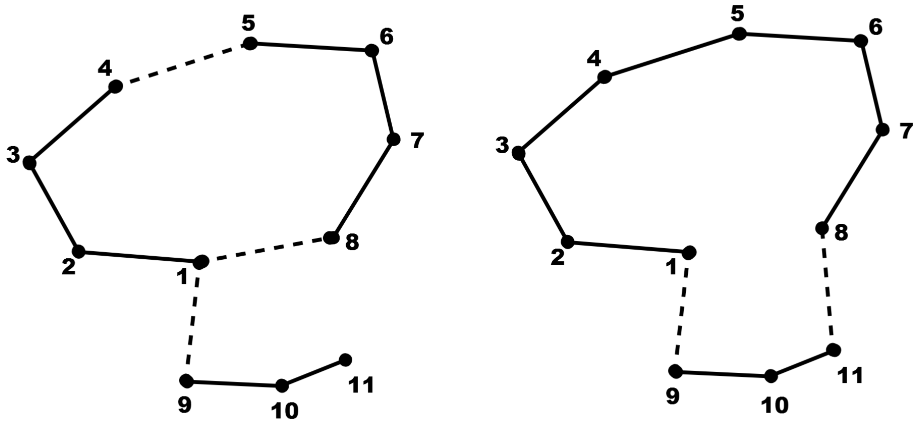

Figure 6.2. Variable Subset Merger TSP example. On the left there are three subsets $S_1 = \{X_1 = 2, X_2 = 3, X_3 = 4, X_4{*}{=}5\}, S_2 = \{X_5 = 6, X_6 = 7, X_7 = 8, X_8{*}{=}1\}, S_3 = \{X_9{*}{=}1, X_{10} = 9, X_{11} = 10\}$ '=' denotes fixed, and '*=' recommended assignments respectively. On the right S_1 and S_2 are merged, variable $X_4 = 5$ is fixed, and new recommendations are calculated for the remaining two free variables X_8 and X_9.

6.1.1.1. VSM-Const

VSM-Const is the simplest way of subset selection. The same subset S_i is selected during the whole process. Only one subset solution is extended to a complete solution. In this case VSM is similar to the subset solving heuristic.

6.1.1.2. VSM-PLoss

When a recommended value of the free variable becomes unavailable, a new recommendation has to be created, which is inferior for the subset. Possible Loss for a subset S_i is the cost difference between the second (r_2) and the first (r_1) recommendation. In Eq. (6.1) f_{S_i} denotes the restricted version of the objective function f, where only $X_l \in S_i$ variables are accounted and X_k is the free variable. VSM-PLoss selects the subset that has the highest possible loss.

$$p\_loss(S_i) = f_{S_i}(X_k = r_2) - f_{S_i}(X_k = r_1) \tag{6.1}$$

VSM-Ploss is strongly connected to the Little branch and bound method [R48].

6.1.1.3. VSM-CLoss

Instead of evading high possible losses, VSM-CLoss minimizes the overall caused loss of a construction step. Caused Loss for subset S_i is the sum of possible losses of all subsets,

DOI:10.15774/PPKE.ITK.2016.007

which recommendations become unavailable if S_i is selected. VSM-CLoss selects the subset that has the lowest caused loss.

$$c\_loss(S_i) = \sum_{S_j \in L} p\_loss(S_j) \tag{6.2}$$

In Eq. (6.2), L is the set of subsets, which suffer their possible losses, if subset S_i is selected. The caused loss is important because this loss surely appears in the final solution's cost. It is possible that the second recommendation also becomes unavailable, therefore Eq. (6.2) is only a lower bound on real caused loss. Calculating caused loss for all subsets is computationally expensive because the next construction step has to be performed virtually to get L.

6.1.1.4. VSM-PCLoss

$$pc\_loss(S_i) = \alpha_{S_i} \cdot c\_loss(S_i) - \beta_{S_i} \cdot p\_loss(S_i) \tag{6.3}$$

The two main subset selection heuristics can be combined with α_{S_i} and β_{S_i} parameters, which can depend on S_i. VSM-PCLoss selects the subset, which has minimal *pc_loss*. Parameters can provide multiple functionalities. For instance set $\alpha_{S_i} = 0$, while β_{S_i} can depend on the size of the subset or the number of constraints. The parameters can be time dependent too. If a subset solution needs to be extended immediately (execution reaches the end of the subset solution), the corresponding β_{S_i} parameter can be set to inf.

6.1.2. VSM Hybrid Methods

6.1.2.1. With best-known not real-time solver

VSM creates an applicable partial solution with appropriate subordinate heuristics, then call the best-known not real-time solver for the rest of the problem.

6.1.2.2. With best-known real-time solver

With best-known real-time solver, VSM creates a candidate solution in constrained time, and the first part of the candidate solution is given as the applicable partial solution. In the next stage a more complex method, for instance, a population-based solver searches for better solutions on the free decision variables. Elements of the candidate solution are

DOI:10.15774/PPKE.ITK.2016.007

seen as recommendations for the VSM process, which accepts them according to its subset selection mechanism. Clearly, the resulting solution will be better than the best solution can be obtained with a single real-time method while the response of the optimization process is still real-time.

6.1.3. VSM models for specific optimization problems

When VSM is applied to a particular problem, first the decision variables have to be defined, and then a heuristic is required, which provides their importance order at each construction step (subset selection). A known greedy method is sufficient to use for the subset recommendations (subset solver). If the distribution of decision variables affect the subset solver or the subset solutions have applicable meaning (TSP subtours), the merger method has to be defined accordingly.
When the feasibility is a hard constraint, the recommendations of subsets calculated by the subset solver heuristic have to be validated. A recommended solution element is valid if its fixation does not prevent feasibility.

6.2. VSM for the Disc Scheduling Problem (DSP)

Scheduling of data transfers for a hard disc drive is a combinatorial optimization problem which requires fast response time. A disk request T_i is given by its ready time r_i, deadline time d_i sector number l_i, data size b_i, and track location a_i. A transfer can start after its ready time and must be finished before the corresponding deadline time. The time need of a transfer T_i which succeeds T_j is given by Eq. (6.4)

$$c_{j,i} = seek\_time(a_j - a_i) + rotational\_latency(l_i) + transfer\_time(b_i) \tag{6.4}$$

$T_{w(1)}, T_{w(2)}, ..., T_{w(n)}$ is a schedule where $w(i)$ is an index function which describes the ordering of the disc requests. Finish time of request $T_{w(i)}$ is $f_{w(i)} = max\{r_i, f(w(i-1))\} + c_{w(i-1),w(i)}$. A schedule is feasible if $f_{w(i)} \leq d_{w(i)}$ for all requests. Finding a feasible schedule with minimal $f_{w(n)}$ is the goal of disc scheduling.
Traditional DSP solvers are fast greedy heuristics such as Shortest Seek-Time First (SSTF), Earliest Deadline First (EDF), SCAN and their modifications. Better results can be obtained by population based more complex heuristics [R49], but their time

DOI:10.15774/PPKE.ITK.2016.007

complexity is too high compared to the constructive methods.

6.2.1. VSM model for the Disc Scheduling Problem

In the simplest case, X_i decision variables define the successor of a request in the schedule. SSTF, EDF and other constructive greedy DSP solvers are special VSM heuristics, where VSM_Const subset selection is used, and the subset solving heuristic is the corresponding greedy method itself.

With the VSM concept, more sophisticated construction methods can be defined. Let X_i denote the predecessor of request T_i. A particular schedule is $Part_m = T_{w_m(1)}, T_{w_m(2)}, ..., T_{w_m(last)}$. The free decision variable of $Part_m$ is the predecessor of its first task. The approximate finish time $f(Part_m)$ for $Part_m$ can be calculated based on its members. If the finish time of the chosen predecessor is greater than the ready time of the particular schedule of $Part_m$, the whole schedule is delayed by that difference. The time pool of a part is the maximal delay which does not cause deadline time conflict in $Part_m$. $Pool_m = \min_{T_k \in Part_m}\{d_k - f_k\}$.

In a VSM solver, SSTF can be used as subset solver heuristic to define $T_{w_m(0)}$ for all $Part_m$ subsets. The subset selection heuristic can deal with deadline time conflicts. That subset is chosen which has minimal time pool. Subset solver heuristic can be sensitive to the deadlines too, if the approximate finish times of the candidates are also accounted with the seek times. Subset selection can also focus on finish time, if that subset is chosen which has the latest approximate finish time.

I do not want to give a concrete method for DSP here, my goal is to show the possibilities and methods of the generation of VSM-based heuristics. First, the decision variables have to be defined, which declare the attributes of subsets: partial objective function ($f(Part_m)$), and measurable other factors ($Pool_m$). Based on the previous step the subordinate subset solver and subset selection heuristics can be defined.

A question arises. How can we guarantee the applicability of partial solutions? Validation of recommendations is required in applicable partial solution generation to ensure applicability. DSP heuristic solutions often violate the deadline constraints, thus I do not suggest validation of recommendations in DSP. Example of validation is presented for the Sequential Ordering Problem (SOP) in the next section.

DOI:10.15774/PPKE.ITK.2016.007

6.3. VSM for the Generalized Assignment Problem

In Sec. (6.1) VSM is presented with a TSP example, which has permutation-based search space representation. Here I apply VSM for a not permutation-based problem through the steps of Sec. (6.1.3).

6.3.1. Generalized Assignment Problem (GAP)

Given n jobs $J = \{1...n\}$ and m agents $I = \{1...m\}$. The assignment $x_{ij} \in \{0,1\}$ of jobs $j \in J$ to agents $i \in I$ is required. Each assignment has c_{ij} cost and a_{ij} resource need from agent i which agent has b_i resource capacity.

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in I} \sum_{j \in J} x_{ij} \cdot c_{ij} \\ \text{subject to} \quad & \sum_{j \in J} x_{ij} \cdot a_{ij} \leq b_i \quad \forall i \in I, \\ & \sum_{i \in I} x_{ij} = 1 \quad \forall j \in J, \\ & x_{ij} \in \{0,1\} \quad \forall i \in I \,\text{and}\, \forall j \in J, \end{aligned} \tag{6.5}$$

GAP is known to be $\mathcal{NP}$-hard (e.g.,[R50]), thus metaheuristics paly important role in handling such problems. The best-known method in solution quality is PREC [R51] a path relinking approach with ejection chains.

6.3.2. VSM model for GAP

There are existing GAP solvers that use variable ordering approach to reach fast running times in the case of complete solution generation. These methods can be redefined with the VSM nomenclature to produce applicable partial solutions. Here I redefine the approach of Martello and Toth [R52] as a VSM.

Each job has a decision variable that is the index of the assigned machine. The subset selection heuristic is VSM-PLoss. Examples of subset solver heuristics used by Martello and Toth are:

(i) minimize c_{ij}

(ii) minimize a_{ij}

DOI:10.15774/PPKE.ITK.2016.007

(iii) minimize a_{ij}/b_i

The first (i) subset solver aims for cost minimization only, the heuristics (ii) and (iii) focus on absolute and relative resource optimization. In [R53] the authors attempt to join the two goals with the family of the following subset solvers:

$$\text{minimize} \quad c_{ij} + \lambda \cdot a_{ij}$$

The distribution of decision variables does not affect these subset solvers and has no additional meaning for subset solutions, therefore subset selection can give back an arbitrary subset.

These methods may produce an infeasible solution, which has to be considered when a partial solution is applied. Unfortunately, validation of an assignment cannot be efficient for GAP because the problem of judging the existence of a feasible solution for GAP is $\mathcal{NP}$-complete. Without effective validation the constraints have to be soft constraints because feasibility can not be guaranteed.

Imagine a server grid and a large bunch of processes. An assignment is required in which all processes are done before a deadline. A VSM-based solver can assign almost immediately one job to all machines according to its subordinate heuristics and assigned processes can be initiated. However, some processes may be assigned to a machine, which has not enough free capacity to handle them without deadline conflict. The probability of such infeasible assignments can be decreased by penalty functions, but could not be zero. Without VSM, one can sacrifice predefined amount of time to generate a complete solution with good quality. However, the machines stay idle during the optimization while the deadline is the same.

In the next section I show a constrained combinatorial optimization problem, where validation of solution elements can be performed, therefore hard constraints are allowed for VSM.

DOI:10.15774/PPKE.ITK.2016.007

6.4. VSM for the Sequential Ordering Problem

6.4.1. Sequential Ordering Problem (SOP)

SOP is a combinatorial optimization task, which was defined by Escudero [R54]. SOP can be derived from the TSP by adding precedence constraints to the cities. Every city can have a list of other nodes, which have to precede them in the solution. Since all TSP instances are SOP ones without ordering, SOP is also $\mathcal{NP}$-hard. Many real-world tasks lead to SOP: vehicle routing with pick-up and delivery tasks, production planning, transportation problems in flexible manufacturing systems, robot action planning. Exact algorithms were developed by Ashauer and Escudero [R55, R56]. A sophisticated branch and bound method is also introduced in [R57]. Several metaheuristics were published to solve SOP: a genetic algorithm based on Voronoi quantized crossover (VGA) by Seo [R58], an ant-colony system (ACS) by Gambardella [R59, R60] where a new 3-Excange method is also presented and a discrete particle swarm optimization (DPSO) by Anghinolfi [R61]. The best-known method is an enchanted ant-colony system (EACS)[R62].
SOP is a challenging CO problem with constraints. The formal definition is as follows: Given a directed complete graph $G(V, E)$ and a weight-function on the edges $c_{i,j} = c(e_{v_i,v_j})$ $v_i, v_j \in V$ $e \in E$. Furthermore, a precedence digraph $P(V, R)$ with no directed cycles is also given, where R holds precedence constraints. $e_{v_i,v_j} = (v_i, v_j) \in R$ means that v_i has to precede v_j in every feasible solution.

6.4.2. VSM-SOP

VSM-SOP is a concrete algorithm that is created for the investigation of multiple subset solution extension in a constrained problem. The satisfiability of constraints defined by P is preserved during the construction. Satisfiability is ensured by compulsory validations of the recommendations.
Decision variables X_i define the successors of nodes v_i. R is contradiction-free and transitively closed. The subsets partial objective function is the length of subtours. VSM-SOP uses the Nearest Neighbor Heuristic for creating the recommendations. A validated v_{r1} is recommended for variable X_i, for which $c_{i,r1}$ is minimal.
Subset selection is performed by the VSM-PLoss heuristic. $p\_loss(S_i) = c_{k,r2} - c_{k,r1}$, where

DOI:10.15774/PPKE.ITK.2016.007

X_k is the free variable in S_i, and v_{r1},v_{r2} are the first and second validated recommendations. If only one recommendation is valid for a variable, $p\_loss(S_i) = inf$. If more than one of the subsets have the maximal possible loss, the method chooses the subset with the lowest index. Because of the reserved satisfiability, if a subset has no valid recommendation the corresponding subset solution surely will be the last element in the ordering (has no successor).
In the case of SOP, decision variables define the subset S_j for the merger. S_j is the subset that contains v_{r1}.

6.4.3. Compulsory Validations

The output of VSM-SOP is a feasible solution, in which all constraints are satisfied. It is achieved by compulsory validations of recommended solution elements, which preserve resolvability of the constraints.

Algorithm 7 Compulsory Validations for VSM SOP

```
Precondition: $S$ denotes a subset, $X$ a decision variable $v$: an SOP node AND $v_k \in S_i$
    iff $X_k \in S_i$ AND $v_r$ is the recommended value for the free variable of $S_i$ AND $v_r \in S_r$

1: function VALIDATION 1$(S_i, v_r)$
2:     if $\exists v_k \in S_i$, $\exists v_m \in S_r : (v_m, v_k) \in R$ then
3:         $v_r$ is unavailable for $S_i$
4: function VALIDATION 2$(S_i, v_r)$
5:     if $\exists v_k \in S_i$, $\exists v_m \in S_r$, $\exists v_l \notin S_i \cup S_r : (v_k, v_l) \in R$ and $(v_l, v_m) \in R$ then
6:         $v_r$ is unavailable for $S_i$
7: function VALIDATION 3$(S_i, v_r)$
8:     if $\exists v_k \in S_i$, $\exists v_m \in S_r$, $\exists v_l \notin S_i \cup S_r : (v_l, v_k) \in R$ and $(v_m, v_l) \in R$ then
9:         $v_r$ is unavailable for $S_i$
```

Subset solutions in VSM-SOP are ordered sets of nodes, where the successor of the last node is the open variable of the subset. The recommended successor v_r is the starting node of S_r. It is obvious, if a node in S_r has to precede a node that is in S_i, the recommendation of v_r directly violates the constraints (Validation 1). If a node exists that has to be placed between S_i and S_r, the merge of the two subset solutions will lead to unsatisfiability (Validation 2). It is also impracticable if a node has to be before and after the merged subset solutions at the same time (Validation 3). All of the recommendations must pass the three compulsory tests in VSM-SOP. The high computational complexity of these validations can be decreased by defining relations between the subsets instead

DOI:10.15774/PPKE.ITK.2016.007

of the nodes. However, relations that defined on subsets have to be updated in every construction step.

6.5. Results

6.5.1. Subset selection and optimality

Recent papers on SOP [R61, R62] use the SOPLIB benchmark, which is available online[1]. The instances in SOPLIB are denoted as $R.n.r.p$, where n is the number of nodes, r is the cost range, and p is the percentage of precedence constraints. TSPLIB is an old problem set[2], where the optimal solution has been already known for many instances[3]. It involves real-life problems for example (rbgxxxa), which derives from a stacker crane application, and randomly generated instances (prob.x). Problems (ftxx.x and kroxxxp.x) have been generated starting from asymmetric traveling salesman problem instances by adding precedence constraints. The comparison of different methods is based on the published results on the same problem instances. However, the authors used different processor architectures for the evaluation, thus the difference between the processors must be considered. To make it easier, I present single thread results on my Core 2 Duo processor.

EACS is the state-of-the-art method for SOP, but no running time information is published, only the 600 sec bound. DPSO results are close to EACS and running time information is available, therefore I use DPSO as reference solver in the later.

DPSO has 3 weighting parameters: reservation of previous particle velocity (inertia parameter), moving towards the local best solution of the particle (cognitive parameter) and moving towards the best-known solution of all particles (social parameter). The authors of [R61] explored many parameter configurations and also proposed a parameter adaptation and stagnation avoidance mechanism. In the comparisons, I use the best achieved results of DPSO during the experimental campaign of parameter tuning, with the published average running times.

The investigated VSM variants for the SOP use the Nearest Neighbor Heuristic as subset solver. VSM-CONST represents the class of simple greedy constructive methods, where

[1]http://www.idsia.ch/∼roberto/SOPLIB06.zip

[2]http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/sop/

[3]known solution cost is equal to a lower bound

DOI:10.15774/PPKE.ITK.2016.007

always the same subset is chosen in the VSM procedure, thus VSM-CONST is equal to the subset solver heuristic. VSM-SOP is the proposed VSM-based solver for the SOP, which uses the PLoss subset selection heuristic. VSM-SOP is a deterministic procedure which provides the same output for the same input. VSM-CONST provides different solutions based on the selection of the first node, thus the average and best results of 10 runs are used.

Table (6.1) shows the results for TSPLIB instances. On this problem set VSM-SOP provides average 7.5% better solutions than VSM-CONST, with 29% average distance from best known solutions. Best VSM-CONST solutions from 10 runs are also outperformed by 3.2%. Best solutions can be obtained by iterative population based methods like VGA [67] and DPSO [R61], which use iterative local search for improving their results. Furthermore, these methods are not deterministic, solution costs have 1-10% dispersion. VGA has extremely slow convergence on larger problems, DPSO seems fast, but also many times slower than VSM-SOP on TSPLIB problem set. Real-time requirements can be satisfied with VSM-SOP, response times are under 1 second.

The comparison between VSM-SOP complete solutions and best-known solutions can

Table 6.1. Results for the TSPLIB benchmark

Instance	n	$\lvert R \rvert$	VSM-CONST[a]			VSM-SOP[a]			VGA[67][b]		DPSO[R61][c]	
			10 avg	10 best	time(s)	resp(s)	cost	time(s)	cost	time(s)	cost	time(s)
ESC78	80	283	22600	22600	0.155	0.019	22310	0.11	18230	0.91	18230	55
ft70.1	71	17	45599	44646	0.044	0.022	41808	0.20	39313	6.27	39313	98.2
ft70.2	71	48	47774	47503	0.043	0.015	43883	0.07	40419	7.66	40419	62.1
ft70.3	71	215	55438	52067	0.049	0.027	47772	0.07	42535	2.38	42535	54.4
ft70.4	71	1325	62862	62534	0.094	0.033	57148	0.12	53530	3.83	53530	82.5
kro124p.1	101	33	53521	50582	0.405	0.068	51235	0.10	39420	12.92	39420	62.8
kro124p.2	101	68	55492	52707	0.371	0.060	60699	0.11	41336	12.49	41336	60.5
kro124p.3	101	266	73799	63662	0.250	0.092	79250	0.19	49499	12.71	49499	76.2
kro124p.4	101	2305	97863	94675	0.183	0.067	108199	0.30	76103	8.36	76103	143.8
prob.100	100	41	2739	2184	0.146	0.064	3455	0.11	1163	1767	1213	183.8
rbg109a	111	5329	1474	1443	0.394	0.072	1131	0.70	1038	11.88	1038	118.7
rbg150a	152	10334	2215	2168	0.440	0.113	1862	2.25	1750	34.2	1750	176.2
rbg174a	176	13955	2441	2440	0.311	0.171	2168	3.84	2033	85.85	2033	136.1
rbg253a	255	30181	3556	3554	0.943	0.452	3125	17.33	2950	325	2950	98.2
rbg323a	325	48202	4032	4032	0.922	0.937	3333	41.38	3140	2515	3140	214.5
rbg341a	343	54303	3785	3785	1.447	1.146	2984	54.29	2568	10164	2570	308.9
rbg358a	360	56536	4112	4108	1.306	1.085	3152	56.82	2545	24340	2550	368
rbg378a	380	63585	4082	4079	1.571	1.392	3287	71.50	2816	33774	2817	395.4

a: Core 2 Duo 2.26 GHz (single thread)
b: Pentium III. 1.1 GHz
c: AMD Opteron250 2.4 GHz
n: number of nodes, $|R|$: number of precedence constraints

demonstrate the scale of optimality loss caused by fixed partial solutions. Table 6.2 shows

DOI:10.15774/PPKE.ITK.2016.007

Table 6.2. Optimality of different subset selection methods, EACS and DPSO on the SOPLIB benchmark

Instance	RND10	CONST10	PLOSS	CLOSS	PC-LOSS	DPSO_Best[R61]	EACS10[R62]	PLOSS Time (s)[b]	DPSO Time (s)[a]
R.200.100.1	462.7	384	**142**	400	345	64	67.2	0.03	224.3
R.200.100.15	5096.2	4062	3866	4133	**3809**	1796	1818.3	0.80	228.6
R.200.100.60	82781.7	78198	73400	75925	**72890**	71749	71749	1.30	18.6
R.200.1000.1	5120.1	4554	**2149**	5022	5022	1414	1432.5	0.03	303.9
R.200.1000.15	50084.9	41323	33260	42634	**32758**	20481	20717	0.86	200.3
R.200.1000.60	83716.4	84431	76850	76700	**76039**	71556	71556	1.35	112.6
R.400.100.1	463.2	351	**191**	285	285	21	22.6	0.13	411.6
R.400.100.15	10137.1	8899	7464	7459	**7370**	3946	3986.2	20.47	462.7
R.400.100.60	17290.8	17376	**15856**	16404	15930	15228	15228	26.96	310.5
R.400.1000.1	5945.9	5226	**2180**	4437	4437	1484	1475.5	0.14	518.5
R.400.1000.15	99890.1	83803	79494	**76766**	77026	40054	40122.9	21.43	493.3
R.400.1000.60	165258	159888	**145432**	149359	147390	140816	140816	27.71	290.3
R.600.100.1	463.7	424	143	378	**114**	11	9.4	0.31	449.4
R.600.100.15	14530.9	13120	**10584**	11617	11472	5923	5881.7	134.50	573.7
R.600.100.60	26427.8	26661	24498	24418	**24079**	23259	23293	170.52	400.3
R.600.1000.1	6147.5	5675	**3009**	4931	4931	1628	1598.9	0.31	499.1
R.600.1000.15	148322	121845	118660	120242	**108118**	59177	58281.6	125.87	572.5
R.600.1000.60	247309	255691	229523	232013	**227681**	214608	214608	169.16	384.2
R.700.100.1	490.8	442	120	446	**93**	9	7.9	0.44	413.0
R.700.100.15	17968.7	14564	13679	14527	**13146**	7719	7444	267.51	582.1
R.700.100.60	28017.8	27107	24998	26006	**24943**	24116	24172	347.69	366.2
R.700.1000.1	6236.9	5326	**2243**	4886	4886	1606	1614.8	0.48	429.0
R.700.1000.15	175869	150198	**119095**	134316	126182	72618	68630	265.30	575.3
R.700.1000.60	284384	277559	256211	260507	**253260**	245589	245684	325.33	364.4

Bold entries indicate the best VSM-based results
a: AMD Opteron250 2.4 GHz
b: Core i7 3630QM 3.4 GHz (single thread)

the complete solution costs of different subset selection methods. RND10 is the average of 10 runs with random subset selection, CONST10 and EACS10 are 10 average results respectively. Nearest neighbor is used as subset solver heuristic for all VSM methods. Complete solutions generated by PLoss surpass the CONST selection for all instances, which indicates that subset selection can increase the power of the subset solver. Percentage differences between the CONST and Ploss subset selection methods are presented in Fig. (6.3). The benefits of Ploss subset selection for the SOP prove that the possibility of subset selection improves the solutions of the subproblem solving heuristic, especially for the unconstrained problem instances.

According to Table 6.2 optimality loss can be significant, but it decreases with the size of the instances and the number of constraints. The population-based DPSO method may be weaker for these instances because the population generation does not consider the constraints directly. The results of complete VSM-only solutions are just upper bounds on the solution cost in the case of hybridization, where for instance EACS can be used for

DOI:10.15774/PPKE.ITK.2016.007

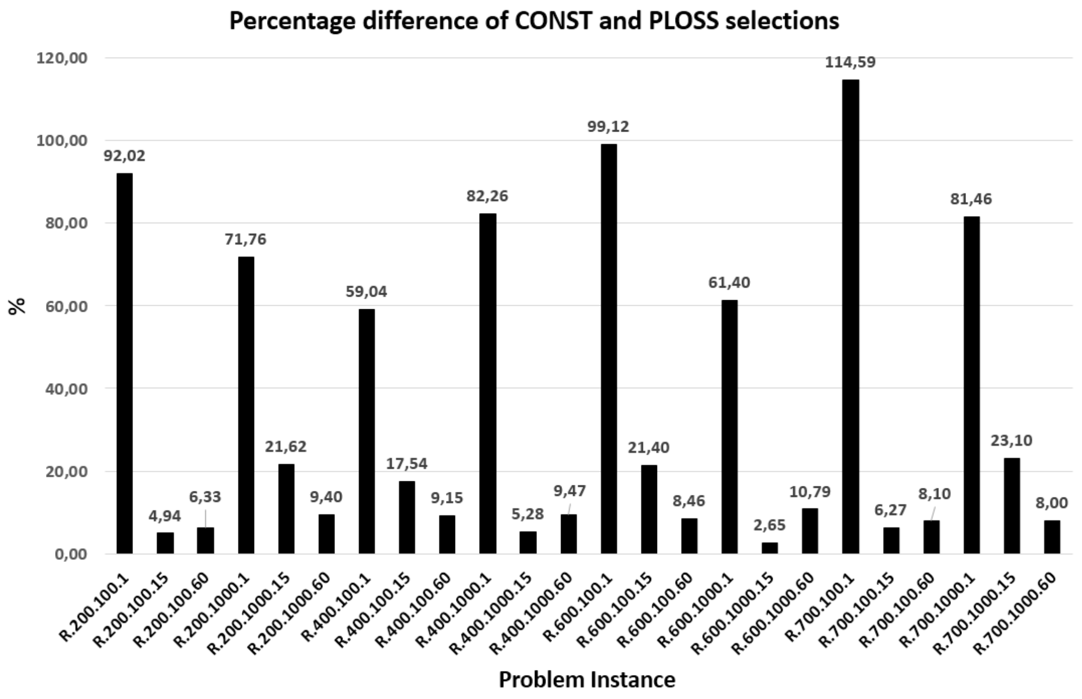

Figure 6.3. Positive effect of subset selection on the performance of the subset solver. Percentage difference $= 100 * (A - B)/avg(A, B)$ %

completing the fixed partial solution.

The validation process is applied for the recommendations in each step, which can affect the running time significantly. For large instances, complete solution time reaches the time complexity of population-based hybrid methods Table 6.2. The fastest selection method is the CONST because validations and preparations have to be done for only one subset. The number of selectable subsets can be decreased for PLoss and CLoss to reach better running time, but it affects the quality of the results.

6.5.2. Response time

Response time ($resp_1$) indicates when the first applicable solution element is created. It includes the necessary initializations, therefore the second solution element can be computed several times faster $resp_2 - resp_1 \ll resp_1$ where $resp_k$ is the time need for an applicable partial solution with k elements. Figure 6.4 shows the time evolution of the partial solution for the R.700.1000.60 problem instance. In the case of not VSM-based solvers, the response time is equal to the complete solution time $resp_1 = resp_n$ because all solution elements can be changed during the optimization.

DOI:10.15774/PPKE.ITK.2016.007

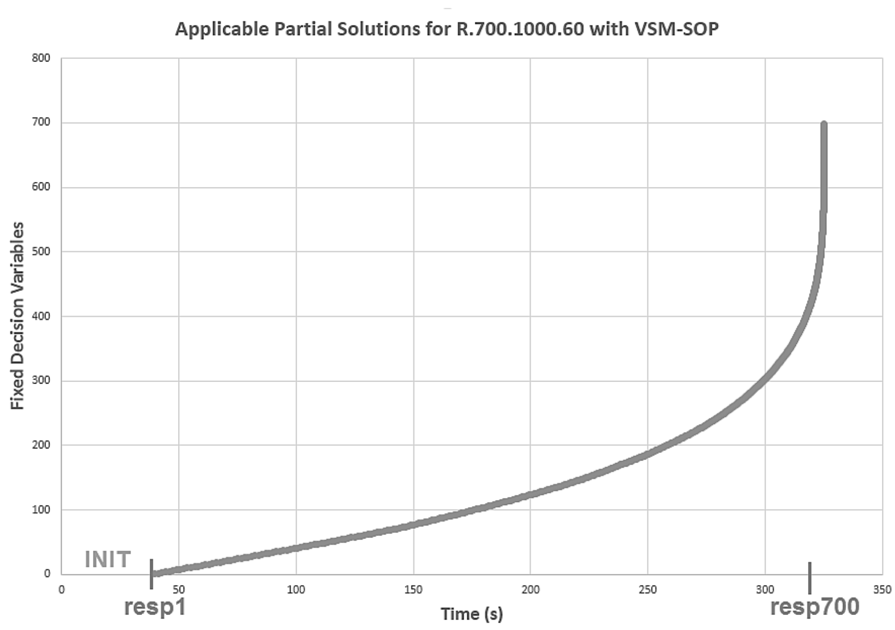

Figure 6.4. Number of fixed decision variables over time. The first decision variable is ready at resp1=42.013 sec which includes initialization. The next 100 variables fixed in 100 seconds (1 second each), then the process is getting faster with the decreased number of free decision variables.

Initialization of VSM-SOP includes the input parsing from a text file, the ordering of possible connections for all subsets, and the validation of the first and second recommendations for all subsets. These steps can be time-consuming for large instances. Response times can be further decreased with parallelization and the restriction of the number of selectable subsets.

6.5.3. Operations with optimization time complexity

VSM is presented to give applicable partial solutions fast in a hybrid optimizer. However, there are situations where a VSM-based method can be the most effective solver alone. If the objective function is time, the optimizer running time becomes an additional cost. For the largest R.700.1000.60 instance VSM-SOP provides a solution with only 5% cost difference from the best available solution obtained in 364 seconds by DPSO. Assuming a 700 seconds solution with DPSO, it results in a 735 seconds solution with VSM-SOP, while the response time is 320 seconds faster, thus the overall time of operations with optimization would be 285 seconds faster. It means more than 25% speedup and 9 times faster response.

Assume that SOPLIB instances describe time minimization problems. Let C denote the

DOI:10.15774/PPKE.ITK.2016.007

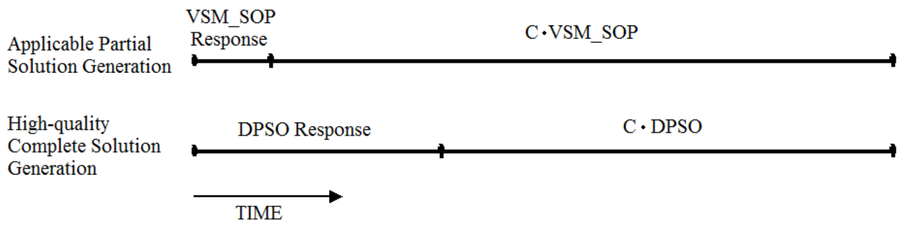

Figure 6.5. Comparison of operations with optimization performance on SOPLIB instances. Time exchange factor C is defined at the point where the two approaches have similar overall time need.

exchange factor between the costs of SOPLIB solutions and execution times of the corresponding rountrips[1]. In Eq. (6.6) C is given at the point where operations with optimization cost is equal for VSM-SOP and DPSO (Fig. 6.5). With this C the average operation time cost can be obtained. If the average operation time cost is smaller than *oper_avg* in Eq. (6.6), VSM-SOP provides better operations with optimization time cost.

$$\begin{aligned}&(DPSO\_resp1 - VSM\_SOP\_resp1) = (VSM\_SOP - DPSO) \cdot C \\ &oper\_avg = \frac{1}{n} \cdot C \cdot DPSO\end{aligned} \tag{6.6}$$

Table 6.3 presents the bounds of average operation time cost for the investigated SOPLIB instances. Bounds vary from 0.05 to 21.7 seconds and always greater than 0.9 sec for constrained instances. In a robotic application assuming a fast actuator, VSM-SOP can be sufficient to handle SOP without hybridization. Conversely in a logistic routing task, these average operation time cost bounds are extremely small.

6.6. Conclusions

In this chapter, Variable Subset Merger a general framework for applicable partial solution generation is presented. The basic possible subordinate heuristics are discussed. Algorithm development examples are shown for GAP and SOP.

VSM can be used for optimization problems, where a partial solution has utilizable meaning, and response time of the optimizer is important. If the existence of a feasible solution cannot be efficiently checked (GAP example), the constraints must be seen as soft constraints. Otherwise, validation procedures can be defined to ensure feasibility (SOP example). The compulsory validation of an intermediate partial solution is the only factor that limits the usage of applicable partial solution generation, and this limitation

[1]SOP is an asymmetric TSP with precedence constraints

DOI:10.15774/PPKE.ITK.2016.007

Table 6.3. Operations with optimization time complexity analysis

Instance	DPSO[R61]	VSM SOP	DPSO[R61] resp1[a] (s)	VSM-SOP resp1[b] (s)	oper_avg* bound (s)
R.200.100.1	64	142	224.3	0.0351	0.9200
R.200.100.15	1796	3866	228.6	0.0823	0.9911
R.200.100.60	71749	73400	18.6	0.2788	3.9557
R.200.1000.1	1414	2149	303.9	0.0332	2.9227
R.200.1000.15	20481	33260	200.3	0.0862	1.6041
R.200.1000.60	71556	76850	112.6	0.2723	7.5836
R.400.100.1	21	191	411.6	0.1284	0.1270
R.400.100.15	3946	7464	462.7	0.8938	1.2939
R.400.100.60	15228	15856	310.5	3.8903	18.4886
R.400.1000.1	1484	2180	518.5	0.1347	2.7628
R.400.1000.15	40054	79494	493.3	0.8964	1.2492
R.400.1000.60	140816	145432	290.3	3.7833	21.7309
R.600.100.1	11	143	449.4	0.2878	0.0623
R.600.100.15	5923	10584	573.7	4.7716	1.2007
R.600.100.60	23259	24498	400.3	22.1376	11.5428
R.600.1000.1	1628	3009	499.1	0.2996	0.9797
R.600.1000.15	59177	118660	572.5	4.4375	0.9388
R.600.1000.60	214608	229523	384.2	20.3528	8.5219
R.700.100.1	9	120	413	0.3724	0.0477
R.700.100.15	7719	13679	582.1	9.3776	1.0524
R.700.100.60	24116	24998	366.2	42.2405	11.9660
R.700.1000.1	1606	2243	429	0.4084	1.5430
R.700.1000.15	72618	119095	575.3	9.1183	1.2552
R.700.1000.60	245589	256211	364.4	42.0132	10.0696

a: AMD Opteron250 2.4 GHz
b: Core i7 3630QM 3.4 GHz (single thread)
*:In Eq. (7.7) VSM_SOP_resp1 is multiplied by 1.417
to consider different processors.

is independent of VSM.

Experimental analysis of the SOP solvers confirmed that the usage of subset selection can increase the performance of the subset solving heuristic. VSM-SOP alone is not sufficient to create complete solutions for the SOP. However, in special applications where SOP defines a time minimization problem, and the average time cost of decisions is relatively small, VSM-SOP provides the best overall time performance. The results on SOPLIB benchmark indicate that VSM can support real-time handling of large combinatorial optimization problems.

The metaheuristic formulation makes the hybridization easy with the best-known real-time and not real-time methods. The solutions of best real-time heuristics can be further optimized with the same response time and VSM makes the use of not real-time heuristics possible in real-time systems.

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

7. Chapter

Theses of the Dissertation

This chapter gives a concise summary of the main scientific contributions of this dissertation as well as the methods and tools used, and briefly discusses the applicability of the results.

7.1. Methods and tools

Most of the experiments were implemented in own-written C++ environments which were created in Microsoft Visual Studio 2008/2012 development tools. Matlab 2008 was also applied for developing test scripts. Graphical User Interfaces were developed with Windows Forms technology. Standard file formats (.msh) were used for data transfers between the own-written and third party software elements. Gmsh [R44], an open-source mesh generation and partitioner tool was used for mesh generation and visualization of the results. I chose METIS [R42] and SCOTCH [R37] for deeper analysis from the available mesh partitioning tools METIS, CHACO, SCOTCH, JOSTLE, and PARTY [R63]. I used METIS as reference solver in performance comparisons.
A dataflow architecture for unstructured mesh computations was analysed on an Alpha-Data ADM-XRC-6T1 reconfigurable development system. Further memory interface tests were made on a Xilinx Zedboard developer board, which consists a Xilinx Zinq SoC processor. This environment has an embedded FPGA, which was programmed with the Vivado HLS 14.4 toolchain.
The performance comparison of the generalized method for applicable partial solution generation was evaluated on two scientific benchmarks TSPlib [R64] and SOPlib [R65].

DOI:10.15774/PPKE.ITK.2016.007

DPSO [R61] was used as reference solver which is one of the best available metaheuristic solutions for SOP.
During my research, I made most of the time experimental analysis but for some special cases, I have proved some simple theorems too.

7.2. New scientific results

1. Thesis Group: Joint handling of memory access and inter-processor communication in mesh partitioning.

Related publications: [J1, C2, C1, C4, C5]
The available mesh partitioning tools do not consider data locality of the output submeshes directly, however, this is crucial in memory access optimization. Partitioning packages provide data locality improvement inside the submeshes. The achievable data locality is defined by the submeshes thus partitioning has major effect on data locality.
My goal is to create a partitioning scheme in which a limit can be defined on data locality which limit depends on the physical parameters of a processor architecture (on-chip memory size). This requirement appeared in the FPGA realizations of Dataflow Machines. These architectures were found to be the most power-efficient in the case of explicit numerical approximation of partial differential equations [R13, J1]. The success of these processors is based on the perfect caching mechanism which works with zero cache-miss. However, the input data stream can not exceed a data locality limit. Partitions which are optimized for inter-processor communication often have to be modified to fit the data locality limit, which modifications degrade the quality of inter-processor communication. The connections between the two optimization goals become important to analyze.

1.1. I showed that the minimization of inter-processor communication and the maximization of data locality are conflicting goals, thus, data locality also must be considered in mesh partitioning. I have experimentally proved that with minimal inter-processor communication growth (<1%), data locality can be increased significantly (>30%).
The optimal ordering of the mesh is necessary for the determination of graph bandwidth which is the indicator of data locality. The corresponding problem is known to be NP-hard,

DOI:10.15774/PPKE.ITK.2016.007

so there is no efficient way to generate the optimal ordering. Empirical and theoretical investigations gave some relations which make the comparison of graph bandwidth of graphs possible. The basic connections were already known, for instance, the connection between the graph diameter and graph bandwidth on which the CM [R10] and GPS [R11] methods are grounded. On the side of inter-processor communication, it was known that the good partitions consist sphere-like submeshes because this geometry has the smallest surface (edge-cut). I have just connected the results of the two optimization problems.
Data locality can be improved optionally with the increase of inter-processor communication. If we have more processors than the chromatic number of the graph, and the graph is partitioned according to the color classes, all data dependencies become inter-processor communication. Conversely, the inter-processor communication can not be decreased under a limit (minimal edge-cut partitioning) with more processors or with worse data locality. Here, the extreme case is when only one processor takes part in the computation, and there is no inter-processor communication. Experimental results showed that partitioning has a major impact on data locality. With the same number of processors for multiple mesh instances, only 0,002 growth in the communication to computation ratio was enough to increase data locality by 30-40%.

1.2. I experimentally showed that if the boundary node set of the mesh is known, the deepest levels of the breadth-first search tree started from this set defines a critical region which has to be cut by a separator to reach better data locality. I also gave a partitioning method which cuts these regions and reaches 30-40% better data locality than METIS in the case of bipartitioning at the expense of inter-processor communication.
In many applications like spatial meshes of PDE approximations, the boundary node set is known (boundary conditions). If a breadth-first search is started from this node set, the search tree defines a depth-level structure (DLS), in which the deepest levels contain nodes which have the largest distance from the surface. DLS structure shows the regions where the center points of the largest spheres reside. The best separators for data locality have to cut these spheres.
The measurement of data locality was performed by the GPS method which is a fast and efficient mesh reordering heuristic.

DOI:10.15774/PPKE.ITK.2016.007

1.3. I gave an extension of the graph partitioning problem (Bandwidth-Limited Partitioning) which consists the joint handling of inter-processor communication and data locality and also optimizes the number of utilized processors. I proved that the solutions of BLP are better for the FPGA-based dataflow architectures than the solutions of the original problem.

In our case, the goal of graph partitioning is to define a distribution of tasks defined by the graph which distribution leads to the fastest running time. For dataflow machines beyond uniform size and minimal edge-cut, more factors have to be considered.

7. Definition (Bandwidth-Limited Partitioning). *Given a graph $G(V, E)$, with vertex set V ($|V| = n$) and edge set E. BW_Bound, $COMM\_Bound$ and K are given parameters. A partition $Q = \{P_1, P_2, .., P_k\}$ is needed which maximizes the number of parts k considering the following conditions:*
Let $Out(P_i)$ denotes the set of outgoing edges of P_i.

$$k \leq K \tag{7.1}$$

$$\max_i \left\{ \frac{|Out(P_i)|}{|P_i|} \right\} \leq COMM\_Bound \tag{7.2}$$

$$\max_i \{2 \cdot B_{f_i}(P_i) + 1\} \leq BW\_Bound \tag{7.3}$$

$$|P_i| \approx \frac{n}{k} \quad \forall i \tag{7.4}$$

Bounds on inter-processor communication (7.2) and data locality (7.3) provide the desired efficiency. Size balance is described by equation (7.4).
Because of the constrained nature of BLP, it is possible that there is no solution. In the case when BLP has no solution, one of the bounds must be relaxed to a higher value.
The limit on communication to computation ratio $COMM\_R$ defines the point where the importance of inter-processor communication and data locality are equal. $COMM\_R$ also limits the number of processors, because the communication to computation ratio becomes larger with more submeshes.

DOI:10.15774/PPKE.ITK.2016.007

1.4. I developed a Gibbs Pole Stockmeyer (GPS)-based algorithm (AM1) which creates bandwidth-limited partitioning without optimization of the processor number (partial BLP).

The method extends the GPS reordering mechanism with a proper graph bandwidth estimation. If the bandwidth need of the already labeled part of the mesh reaches the graph bandwidth limit, the algorithm initiates a new part until the whole mesh is partitioned [J1].

1.5. I gave a near-optimal complete BLP solution for all rectangular 2-3D structured grids with grid-type BLP partitioning.

The experiences with rectangular structured meshes gave deeper understanding of the BLP task.

Grid-type partition of an $a \times b$ mesh is a $g_a \times g_b$ grid of uniform $\frac{a}{g_a} \times \frac{b}{g_b}$ submeshes. In the case of grid-type partitioning the inequalities of BLP take a verifiable form (7.5), where $s_a = \left\lfloor \frac{a}{g_a} \right\rfloor$, $s_b = \left\lfloor \frac{b}{g_b} \right\rfloor$ and $S_a = \left\lceil \frac{a}{g_a} \right\rceil$, $S_b = \left\lceil \frac{b}{g_b} \right\rceil$ denote the possible sizes of the sides. In $|Out(P_i)|$ only the non-boundary sides are accounted, each side is multiplied by $\{0, 1, 2\}$. Let m_a, m_b denote the maximum side multipliers.

$$\begin{aligned} 2 \cdot \min\{S_a, S_b\} + 1 &\leq BW\_Bound \\ \frac{m_a \cdot s_a + m_b \cdot s_b}{s_a \cdot s_b} &\leq COMM\_Bound \end{aligned} \tag{7.5}$$

Because **ga x gb** has to be smaller than **K**, the number of possible partitions is defined by (7.6) which is small enough to use exhaustive search to find the best grid-type BLP partition. In 3D the method is still viable, because there **ga x gb x gc** has to be smaller than **K** which gives only a harmonic multiplier to (7.6).

$$K - \left\lfloor \frac{K}{2} \right\rfloor + \sum_{i=1}^{\lfloor K/2 \rfloor} \left\lfloor \frac{K}{i} \right\rfloor \tag{7.6}$$

The best grid-type solution is not necessarily optimal. There are counter examples, where **ga** is not the divisor of **a** or **gb** is not the divisor of **b**. In these cases, there is a better BLP partition which is not grid-type.

1.6. I developed a METIS-AM1 hybrid method for unstructured mesh

DOI:10.15774/PPKE.ITK.2016.007

partitioning.

Since the inter-processor communication can not be decreased except with reduced number of processors, it is not surprising that the bound on it limits the number of utilized processors (**k**). Many decades of research has been made to handle inter-processor communication. If a k-way METIS partition does not fit the communication bound, it is not probable that there is a k+1-way partition which fulfills this requirement. The base idea of the METIS-AM1 hybrid is to use METIS to define **k**. Recursive bisections are performed until the bound is not reached, and between the last two steps interval halving determinates the largest **k** for which the k-way METIS partition fits the inter-processor communication bound. For each part in the k-way METIS partition, AM1 is applied.

2. Thesis Group: Applicable Partial Solution Generation for Fast-response Combinatorial Optimization.

Related publications: [C3]

Solution time of a combinatorial optimization task mainly depends on the number of decision variables, in other words, it depends on the dimension of the solution space.

Applicable partial solution generation (APSG) is a possible way for dimension reduction. A partial solution is created as partial output in constrained time. While the partial solution is applied, the optimization of the remaining part is continued. The time need of partial solution generation defines response time of the solver because the utilization of the solution can be started. Because of the $\mathcal{NP}$-hard nature of many CO problems, the most efficient solvers are heuristics which search for solutions in an exponentially growing solution space. The quality of solutions depends on the available time for the search, thus, the response time of the optimizer and solution quality are conflicting. Applicable partial solution generation provides a better trade-off between optimization time and quality because it makes possible to restrict response time only for a subset of decision variables instead of the termination of the whole optimization process when the required response time is reached. Because multi-level methods could provide fast partitioning, the results of this thesis group are not applied to the acceleration of BLP. However, it is useful for other CO applications.

DOI:10.15774/PPKE.ITK.2016.007

2.1 I gave a metaheuristic framework for applicable partial solution generation (Variable Subset Merger - VSM). I showed the advantages and weak points of VSM on the Sequential Ordering Poblem (SOP), the Disk Scheduling Problem (DSP) and the Generalized Assignment Problem (GAP). I gave the basic subordinate heuristics and a hybridization method which improves the solutions of real-time algorithms.

Variable Subset Merger (VSM) describes the extension of multiple subset solutions until they form a complete solution. During the VSM process, each subset has one free decision variable and has fixed assignments for the rest of variables in it.

VSM can be used for optimization problems, where a partial solution has utilizable meaning, and response time of the optimizer is important. If the existence of a feasible solution cannot be efficiently checked (GAP example), the constraints must be seen as soft constraints. Otherwise, validation procedures can be defined to ensure feasibility (SOP example). The compulsory validation of an intermediate partial solution is the only factor that limits the usage of applicable partial solution generation, and this limitation is independent of VSM.

The results on SOPLIB benchmark indicate that VSM can support real-time handling of large combinatorial optimization problems.

The metaheuristic formulation makes the hybridization simple with the best-known real-time and not real-time methods. The solutions of best real-time heuristics can be further optimized with the same response time, and VSM makes the use of not real-time heuristics possible in real-time systems.

2.2 I experimentally proved that the possibility of subset selection improves the solutions of the subproblem solving heuristic. VSM-Ploss subproblem selection outperforms the constant selection with 7.5% on TSPLIB (n=71..280) and 26% on SOPLIB (n=200..700) respectively.

Measurements on SOPLIB instances confirm that VSM-Ploss selection (VSM-SOP) can provide better quality solutions than the VSM-Const. VSM-SOP provides 26% better solutions on these larger instances. In the VSM concept, I want to define good strategies

DOI:10.15774/PPKE.ITK.2016.007

for subproblem selection and decisions, which do not lead to bad quality solutions. Results of VSM-SOP shows that it is possible to create more efficient heuristics for the generation of fixed partial solutions by using VSM.

2.3 I experimentally proved that VSM-SOP is the most efficient solver if SOP represents minimal time task scheduling and the running time of the optimizer is also counted in the cost. For the SOPLIB bechmark VSM-SOP outperforms DPSO if the average time need of a task is less than 1-20 seconds. VSM is presented to give applicable partial solutions fast in a hybrid optimizer. However, there are situations where VSM can provide best solutions alone. If the objective function is time, the optimizer running time becomes an additional cost. For the largest SOPLIB instance with 700 variables, VSM-SOP provides a solution with only 5% cost difference from the best available solution obtained in 364 seconds by DPSO. Assuming a 700 seconds solution with DPSO, it results in a 735 seconds solution with VSM-SOP, while the response time is 320 seconds faster, thus, the overall time of operations with optimization would be 285 seconds faster. It means more than 25% speedup and nine times faster response.

$$\begin{aligned}&(DPSO\_resp1 - VSM\_SOP\_resp1) = (VSM\_SOP - DPSO) \cdot C\\&oper\_avg = \frac{1}{n} \cdot C \cdot DPSO\end{aligned} \tag{7.7}$$

Assume that SOPLIB instances describe time minimization problems. Let C denote

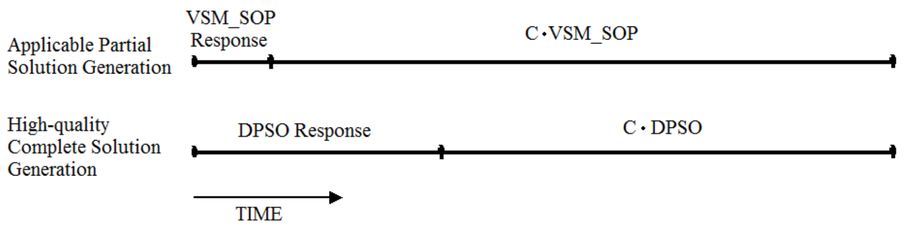
VSM_SOP Response
C•VSM_SOP
Applicable Partial Solution Generation
High-quality Complete Solution Generation
DPSO Response
C•DPSO
TIME

Figure 7.1. Comparison of operations with optimization performance on SOPLIB instances. Time exchange factor C is defined at the point where the two approaches have similar overall time need.

the exchange factor between the costs of SOPLIB solutions and execution times of the corresponding rountrips[1]. In Eq. (7.7) C is given at the point where operations with optimization cost is equal for VSM-SOP and DPSO (Fig. 7.1). With this C the average operation time cost can be obtained. If the average operation time cost is smaller than *oper_avg* in Eq. (7.7), VSM-SOP provides better operations with optimization cost.

[1]SOP is an asymmetric TSP with precedence constraints

DOI:10.15774/PPKE.ITK.2016.007

7.3. Applicability of the results

Results of the first thesis group support the usage of dataflow machines in mesh computing. The AM1 algorithm provides access patterns with constrained data locality. The optimized and bounded access patterns are essential for dataflow machines and enables them to handle larger meshes. AM1 improves the applicability of 1-chip dataflow machines. The second part of the first thesis group provides techniques to create data locality bounded mesh partitioning. Multi-chip dataflow architectures were known for structured grids, but the definition of the corresponding partitioning problem and solvers for the unstructured case were not given earlier.

BLP partitioning is essential for dataflow machines but has an impact on other architectures too when a submesh that is given for one chip is large enough (>300k nodes). For small submeshes, the minimization of inter-processor communication is more important than data locality. However, processor chips have more and more processing capability and off-chip DRAM, which trend makes BLP possibly important for other architectures as well. The results of the first thesis group could also be used for the determination of optimal processor number before partitioning which optimization evades the wasting of resources.

The second thesis group gives methods for response time reduction with applicable partial solution generation in combinatorial optimization. It is useful for CO problems, where a partial solution has utilizable meaning, and response time of the optimizer is important. The metaheuristic formulation makes the hybridization easy with the best-known real-time and not real-time methods. The solutions of best real-time heuristics can be further optimized with the same response time, and VSM makes the use of not real-time heuristics possible in real-time systems. The method without hybridization has been found to be effective for task scheduling when hundreds of short (1-20 sec) tasks with precedence constraints are given.

DOI:10.15774/PPKE.ITK.2016.007

DOI:10.15774/PPKE.ITK.2016.007

References

Author's journal publications

[J1] Nagy, Z. Nemes, C. **Hiba, A.** Csík, Á. Kiss, A. Ruszinkó, M. Szolgay, P. "Accelerating unstructured finite volume computations on field-programmable gate arrays". In: *Concurrency and Computation: Practice and Experience* 26.3 (2014), pp. 615–643.

[J2] Zsedrovits, T. Bauer, P. **Hiba, A.** Nemeth, M. Pencz, B. J. M. Zarandy, A. Vanek, B. Bokor, J. "Performance Analysis of Camera Rotation Estimation Algorithms in Multi-Sensor Fusion for Unmanned Aircraft Attitude Estimation". In: *Journal of Intelligent & Robotic Systems* (2016), pp. 1–19.

[J3] Zsedrovits, T. Bauer, P. Pencz, B. J. M. **Hiba, A.** Gozse, I. Kisantal, M. Nemeth, M. Nagy, Z. Vanek, B. Zarandy, A. Bokor, J. "Onboard Visual Sense and Avoid System for Small Aircraft". In: *IEEE Aerospace and Electronic Systems Magazine (accepted)* (2016).

Author's conference publications

[C1] Nagy, Z. Nemes, C. **Hiba, A.** Kiss, A. Csík, Á. Szolgay, P. "FPGA based acceleration of computational fluid flow simulation on unstructured mesh geometry". In: *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on.* IEEE. 2012, pp. 128–135.

[C2] **Hiba, A.** Nagy, Z. Ruszinko, M. "Memory access optimization for computations on unstructured meshes". In: *Proc. 13th International Workshop on Cellular Nanoscale Networks and their Applications.* 2012.

DOI:10.15774/PPKE.ITK.2016.007

[C3] **Hiba, A.** Ruszinko, M. "Real-time combinatorial optimization with applicable partial solution generation". In: *1st International Conference on Engineering and Applied Sciences Optimization.* 2014, pp. 590–599.

[C4] Nagy, Z. Nemes, C. **Hiba, A.** Kiss, A. Csík, Á. Szolgay, P. "Accelerating Unstructured Finite Volume Solution of 2-D Euler Equations on FPGAs". In: *Conference on Modelling Fluid Flow (CMFF'12).* 2012.

[C5] **Hiba, A.** Nagy, Z. Ruszinkó, M. Szolgay, P. "Data locality-based mesh partitioning methods for dataflow machines". In: *14th International Workshop on Cellular Nanoscale Networks and their Applications.* IEEE, 2014.

[C6] Zsedrovits, T. Zarandy, A. Pencz, B. **Hiba, A.** Nameth, M. Vanek, B. "Distant aircraft detection in sense-and-avoid on kilo-processor architectures". In: *Circuit Theory and Design (ECCTD), 2015 European Conference on.* IEEE. 2015, pp. 1–4.

[C7] Bauer, P. **Hiba, A.** Vanek, B. Zarandy, A. Bokor, J. "Monocular Image-based Time to Collision and Closest Point of Approach Estimation". In: *24th Mediterranean Conference on Control and Automation.* 2016.

[C8] **Hiba, A.** Zsedrovits, T. Bauer, P. Zarandy, A. "Fast horizon detection for airborne visual systems". In: *2016 International Conference on Unmanned Aircraft Systems.* 2016.

[C9] **Hiba, A.** Orzo, L. "Retina simulator challenges, image processing with a varying resolution sensor". In: *15th International Workshop on Cellular Nanoscale Networks and their Applications.* 2016.

[C10] **Hiba, A.** Zarandy, A. Pencz, B. "Remote Aircraft Detection against Sky Background". In: *15th International Workshop on Cellular Nanoscale Networks and their Applications.* 2016.

[C11] Orzo, L. **Hiba, A.** Zarandy, A. "Deconvolution as a model of blur adaptation in the visual cortex". In: *15th International Workshop on Cellular Nanoscale Networks and their Applications.* 2016.

DOI:10.15774/PPKE.ITK.2016.007

Related publications

[R1] Wulf, W. A. McKee, S. A. “Hitting the Memory Wall: Implications of the Obvious”. In: *SIGARCH Comput. Archit. News* 23.1 (Mar. 1995), pp. 20–24. ISSN: 0163-5964. DOI: 10.1145/216585.216588. URL: http://doi.acm.org/10.1145/216585.216588.

[R2] Xie, Y. “Future memory and interconnect technologies”. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 2013, pp. 964–969. DOI: 10.7873/DATE.2013.202.

[R3] Huang, Y.-J. Li, J.-F. “Yield-enhancement Schemes for Multicore Processor and Memory Stacked 3D ICs”. In: *ACM Trans. Embed. Comput. Syst.* 13.3s (Mar. 2014), 106:1–106:22. ISSN: 1539-9087. DOI: 10.1145/2567933. URL: http://doi.acm.org/10.1145/2567933.

[R4] Borkar, S. “Thousand Core Chips: A Technology Perspective”. In: *Proceedings of the 44th Annual Design Automation Conference*. DAC ’07. San Diego, California: ACM, 2007, pp. 746–749. ISBN: 978-1-59593-627-1. DOI: 10.1145/1278480.1278667. URL: http://doi.acm.org/10.1145/1278480.1278667.

[R5] Garey, M. R. Johnson, D. S. *Computers and Intractablility: A Guide to the Theory of NP-completeness*. W. H. Freeman, 1979. ISBN: 0-7167-1044-7.

[R6] Papadimitriou, C. H. “The NP-completeness of the bandwidth minimization problem.” In: *Computing* 16 (1976), pp. 263–270.

[R7] Blum, C. Aguilera, M. Roli, A. Sampels, M. *Hybrid Metaheuristics: An Emerging Approach to Optimization*. Studies in Computational Intelligence. Springer, 2008. ISBN: 9783540782940.

[R8] Blum, C. Puchinger, J. Raidl, G. R. Roli, A. “Hybrid metaheuristics in combinatorial optimization: A survey”. In: *Applied Soft Computing* 11.6 (2011), pp. 4135–4151.

[R9] Karypis, G. Kumar, V. “Multilevel k-way partitioning scheme for irregular graphs”. In: *Journal of Parallel and Distributed Computing* 48.1 (1998), pp. 96–129.

DOI:10.15774/PPKE.ITK.2016.007

[R10] Cuthill, E. McKee, J. “Reducing the bandwidth of sparse symmetric matrices”. In: *Proceedings of the ACM National Conference, Association for Computing Machinery, New York.* 1969, pp. 157–172.

[R11] Gibbs, N. Poole, W. Stockmeyer, P. “An algorithm for reducing the bandwidth and profile of sparse matrix”. In: *SIAM Journal on Numerical Analysis* 13.2 (1976), pp. 236–250.

[R12] Hill, T. “Accelerating Design Productivity with 7 Series FPGAs and DSP Platforms, Xilinx WP406 (v1.1)”. In: 2013.

[R13] Pell, O. Bower, J. Dimond, R. Mencer, O. Flynn, M. J. “Finite-Difference Wave Propagation Modeling on Special-Purpose Dataflow Machines”. In: *Parallel and Distributed Systems, IEEE Transactions on* 24.5 (2013), pp. 906–915. ISSN: 1045-9219. DOI: 10.1109/TPDS.2012.198.

[R14] Nagy, Z. Szolgay, P. Kiss, A. László, E. *Párhuzamos számítógép architektúrák, processzortömbök.* Pázmány Egyetem eKiadó, 2015.

[R15] Kolluri, S. “UltraScale Architecture Low Power Technology Overview, Xilinx WP451 (v1.1)”. In: 2015.

[R16] “Zynq-7000 All Programmable SoC Overview, Xilinx DS190 (v1.9)”. In: 2016.

[R17] Lindtjorn, O. Clapp, R. Pell, O. Fu, H. Flynn, M. Mencer, O. “Beyond traditional microprocessors for geoscience high-performance computing applications”. In: *Ieee Micro* 2 (2011), pp. 41–49.

[R18] Jin, Z. Bakos, J. D. “Extending the BEAGLE library to a multi-FPGA platform”. In: *BMC bioinformatics* 14.1 (2013), p. 25.

[R19] Sykora, J. Kohout, L. Bartosinski, R. Kafka, L. Danek, M. Honzik, P. “The architecture and the technology characterization of an FPGA-based customizable Application-Specific Vector Processor”. In: *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2012 IEEE 15th International Symposium on.* IEEE. 2012, pp. 62–67.

[R20] Pham, P.-H. Jelaca, D. Farabet, C. Martini, B. LeCun, Y. Culurciello, E. “NeuFlow: Dataflow vision processing system-on-a-chip”. In: *Circuits and Systems*

DOI:10.15774/PPKE.ITK.2016.007

(MWSCAS), 2012 IEEE 55th International Midwest Symposium on. IEEE. 2012, pp. 1044–1047.

[R21] Farabet, C. LeCun, Y. Kavukcuoglu, K. Culurciello, E. Martini, B. Akselrod, P. Talay, S. “Large-scale FPGA-based convolutional networks”. In: *Machine Learning on Very Large Data Sets* 1 (2011).

[R22] Giefers, H. Plessl, C. Förstner, J. “Accelerating finite difference time domain simulations with reconfigurable dataflow computers”. In: *ACM SIGARCH Computer Architecture News* 41.5 (2014), pp. 65–70.

[R23] Sato, Y. Inoguchi, Y. Luk, W. Nakamura, T. “Evaluating reconfigurable dataflow computing using the Himeno benchmark”. In: *Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on.* IEEE. 2012, pp. 1–7.

[R24] Nemes, C. Nagy, Z. Szolgay, P. “Efficient mapping of mathematical expressions to fpgas: exploring different design methodologies”. In: *Circuit Theory and Design (ECCTD), 2011 20th European Conference on.* IEEE. 2011, pp. 717–720.

[R25] Ercal, F. Ramanujam, J, Sadayappan, P, “Task allocation onto a hypercube by recursive mincut bipartitioning”. In: *Journal of Parallel and Distributed Computing* 10.1 (1990), pp. 35–44.

[R26] Hammond, S. “Mapping unstructured grid computations to massively parallel computers.” PhD thesis. Rensselaer Polytechnic Institute, Troy, New-York, 1992.

[R27] Pellegrini, F. “Scotch and libScotch 5.1 User’s Guide”. In: 2010.

[R28] Walshaw, C. Cross, M. Everett, G. “Partitioning and mapping of unstructured meshes to parallel machine topologies.” In: *Proc. Irregular’95, number 980 in LNCS.* 1995, pp. 121–126.

[R29] Simon, H. D. Teng, S.-H. “How good is recursive bisection?” In: *SIAM Journal on Scientific Computing* 18.5 (1997), pp. 1436–1445.

[R30] Walshaw, C. Cross, M. “Mesh partitioning: a multilevel balancing and refinement algorithm”. In: *SIAM Journal on Scientific Computing* 22.1 (2000), pp. 63–80.

[R31] Pothen, A. Simon, H. D. Liou, K.-P. “Partitioning sparse matrices with eigenvectors of graphs”. In: *SIAM Journal on Matrix Analysis and Applications* 11.3 (1990), pp. 430–452.

[R32] Hendrickson, B. Leland, R. *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations.* Tech. rep. Sandia National Laboratories, Albuquerque, 1992.

[R33] Barnard, S. T. Simon, H. D. "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems". In: *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing.* 1993, pp. 711–718.

[R34] Fiduccia, C. M. Mattheyses, R. M. "A linear-time heuristic for improving network partitions". In: *Proceedings of the 19th Design Automation Conference.* 1982, pp. 175–181.

[R35] Karypis, G. Kumar, V. "Multilevel algorithms for multi-constraint graph partitioning". In: *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM).* IEEE Computer Society. 1998, pp. 1–13.

[R36] Hendrickson, B. Leland, R. Van Driessche, R. "Skewed graph partitioning". In: *Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing.* 1997.

[R37] Pellegrini, F. "Graph partitioning based methods and tools for scientific computing". In: *Parallel computing* 23.1 (1997), pp. 153–164.

[R38] Dueck, G, Jeffs, J. "A heuristic bandwidth reduction algorithm". In: *Journal of combinatorial mathematics and computers* 18 (1995), pp. 97–108.

[R39] Martı, R. Laguna, M. Glover, F. Campos, V. "Reducing the bandwidth of a sparse matrix with tabu search". In: *European Journal of Operational Research* 135.2 (2001), pp. 450–459.

[R40] Pinana, E. Plana, I. Campos, V. Martı, R. "GRASP and path relinking for the matrix bandwidth minimization". In: *European Journal of Operational Research* 153.1 (2004), pp. 200–210.

[R41] Luo, J. "Algorithms for reducing the bandwidth and profile of a sparse matrix". In: *Computers and Structures* 44 (1992), pp. 535–548.

[R42] Karypis, G. Kumar, V. "A fast and high quality multilevel scheme for partitioning irregular graphs". In: *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392.

DOI:10.15774/PPKE.ITK.2016.007

[R43] WEB, *Alpha-data web size. www.alpha-data.com.*

[R44] Geuzaine, C. Remacle, J.-F. “Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities”. In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331.

[R45] Karypis, G. Kumar, V. “Parallel multilevel series k-way partitioning scheme for irregular graphs”. In: *Siam Review* 41.2 (1999), pp. 278–300.

[R46] LaSalle, D. Karypis, G. “Multi-threaded graph partitioning”. In: *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on.* IEEE. 2013, pp. 225–236.

[R47] LaSalle, D. Patwary, M. M. A. Satish, N. Sundaram, N. Dubey, P. Karypis, G. “Improving graph partitioning for modern graphs and architectures”. In: *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms.* ACM. 2015, p. 14.

[R48] Little, J. D. C. Murty, K. G. Sweeney, D. W. Karel, C. “An Algorithm for the Traveling Salesman Problem”. In: *Operations Research* 11.6 (1963), pp. 972–989.

[R49] Bonyadi, M. R. Rahmani, H. Moghaddam, M. E. “A genetic based disk scheduling method to decrease makespan and missed tasks”. In: *Information Systems* 35.7 (2010), pp. 791 –803. ISSN: 0306-4379. DOI: `http://dx.doi.org/10.1016/j.is.2010.04.002`. URL: `http://www.sciencedirect.com/science/article/pii/S0306437910000281`.

[R50] Sahni, S. Gonzalez, T. “P-Complete Approximation Problems”. In: *J. ACM* 23.3 (July 1976), pp. 555–565. ISSN: 0004-5411. DOI: `10.1145/321958.321975`. URL: `http://doi.acm.org/10.1145/321958.321975`.

[R51] Yagiura, M. Ibaraki, T. Glover, F. “A path relinking approach with ejection chains for the generalized assignment problem”. In: *European journal of operational research* 169.2 (2006), pp. 548–569.

[R52] Martello, S. Toth, P. “An algorithm for the generalized assignment problem”. In: *Operational research* 81 (1981), pp. 589–603.

[R53] Romeijn, H. E. Morales, D. R. “A class of greedy algorithms for the generalized assignment problem”. In: *Discrete Applied Mathematics* 103.1 (2000), pp. 209–235.

DOI:10.15774/PPKE.ITK.2016.007

[R54] Escudero, L. "An inexact algorithm for the sequential ordering problem". In: *European Journal of Operational Research* 37.2 (1988), pp. 236 –249.

[R55] Ascheuer, N. Escudero, L. Grötschel, M. Stoer, M. "A Cutting Plane Approach to the Sequential Ordering Problem (with Applications to Job Scheduling in Manufacturing)". In: *SIAM Journal on Optimization* 3.1 (1993), pp. 25–42.

[R56] Escudero, L. Guignard, M. Malik, K. "A Lagrangian relax-and-cut approach for the sequential ordering problem with precedence relationships". In: *Annals of Operations Research* 50.1 (1994), pp. 219–237. DOI: 10.1007/BF02085641.

[R57] Hernadvolgyi, I. T. "Solving the Sequential Ordering Problem with Automatically Generated Lower Bounds". In: *Operations Research Proceedings*. Vol. 2003. Springer Berlin Heidelberg, 2004, pp. 355–362. ISBN: 978-3-540-21445-8.

[R58] Seo, D.-I. Moon, B.-R. "A Hybrid Genetic Algorithm Based on Complete Graph Representation for the Sequential Ordering Problem". In: *Genetic and Evolutionary Computation — GECCO 2003*. Vol. 2723. Lecture Notes in Computer Science. 2003, pp. 669–680.

[R59] Gambardella, L. M. Dorigo, M. "An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem". In: *INFORMS J. on Computing* 12.3 (2000), pp. 237–255. ISSN: 1526-5528.

[R60] Montemannia, R. Smith, D. Gambardella, L. "A heuristic manipulation technique for the sequential ordering problem". In: *Computers and Operational Research* 35 (2008), pp. 3931–3944.

[R61] Anghinolfi, D. Montemanni, R. Paolucci, M. Gambardella, L. "A hybrid particle swarm optimization approach for the sequential ordering problem". In: *Computers and Operational Research* 38 (2011), pp. 1076–1085.

[R62] Gambardella, L. M. Montemanni, R. Weyland, D. "Coupling ant colony systems with strong local searches". In: *European Journal of Operational Research* 220.3 (2012), pp. 831–843.

[R63] Margoules, F. *Mesh Partitioning Techniques and Domain Decomposition Methods*. Saxe-Coburg Publications, 2007. ISBN: 978-1-874672-29-6.

DOI:10.15774/PPKE.ITK.2016.007

[R64] WEB, *TSPLIB95 SOP problem package, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/sop/*. 2014.

[R65] WEB, *SOPLIB problem package, http://www.idsia.ch/∼roberto/SOPLIB06.zip*. 2014.

[R66] Pinana, E. Plana, I. Campos, V. Marti, R. "GRASP and path relinking for the matrix bandwidth minimization". In: *European Journal of Operational Research* 153.1 (2004), pp. 200–210.