Available online at www.sciencedirect.com

ScienceDirect

Procedia Engineering 61 (2013) 152 – 157

Procedia Engineering

www.elsevier.com/locate/procedia

Parallel Computational Fluid Dynamics Conference (ParCFD2013)

Hybrid OpenMP/AVX Acceleration of a Higher Order Quiet Direct Simulation Method for the Euler Equations

Matthew R. Smith[a]*, Ji-Yueh Liu[a], Fang-An Kuo[b,c] and Jong-Shin Wu[b,c]

[a] *Department of Mechanical Engineering, National Cheng-Kung University, Tainan,Taiwan ROC.*
[b] *National Center for High-performance Computing, HsinChu, Taiwan ROC.*
[c] *Department of Mechanical Engineering, National Chiao-Tung University, HsinChu, Taiwan ROC.*

Abstract

Presented is the Quiet Direct Simulation (QDS) applied to parallel computation using a hybrid OpenMP/AVX parallelization paradigm. Due to the high locality of the QDS scheme, the method has been successfully applied to parallel computation using Graphics Processing Units (GPU) – we show here that the same principles which allow high performance on GPU devices also permit high performance when using Advanced Vector eXtensions (AVX). Furthermore, since modern CPU's employ a large number of cores, we can further extend the performance by using AVX on each available CPU core using shared memory (OpenMP) parallelization. We present a simple direction-split higher order extension to the QDS method, and then apply it to AVX through the use of intrinsic functions in the flux computation and state computation modules. High performance is obtained by ensuring that all flux computations are performed using only AVX intrinsic functions – no computations are performed in serial. Through this approach, a single workstation with 2x Xeon CPU's (16 physical cores) allows a performance increase of over 177 times that of a single core alone. We also demonstrate that built-in optimization does not fully exploit AVX parallelization through the examination of assembly code.

© 2013 The Authors. Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
Selection and peer-review under responsibility of the Hunan University and National Supercomputing Center in Changsha (NSCC)

Keywords: Quiet Direct Simulation, True Direction Equilibrium Flux Method, Kinetic Theory of Gases, High Resolution, Total Variable Diminishing, Euler Equations, OpenMP, AVX, Advanced Vector eXtensions.

Nomenclature

ρ	gas density
u	velocity in the x direction
v	velocity in the y direction
T	equilibrium gas temperature
C_v	specific heat capacity
R	specific gas constant
$H(v)$	Heaviside step function
A	radius of (m)
B	position of
C	further nomenclature continues down the page inside the text box
Greek symbols	
γ	ratio of specific heats
Subscripts	
eff	effective (limited) value

* Corresponding author. Tel: +886-6-275-7575; E-mail address: msmith@mail.ncku.edu.tw

1877-7058 © 2013 The Authors. Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
Selection and peer-review under responsibility of the Hunan University and National Supercomputing Center in Changsha (NSCC)
doi:10.1016/j.proeng.2013.07.108

1. Introduction

The last decade has been a turbulent time in the development of computational technology. This has in turn impacted on the way scientists and engineers perform research, in particular to the way in which we perform computation. Recent years have seen a significant increase in the number of possibilities available for parallelization of our simulation algorithms. We have seen significant amounts of research work in both shared and distributed memory parallel computing, using common tools such as MPI (Message Passing Interface), and with the increasing number of cores per physical CPU, an increase in the use of OpenMP. The increase in popularity of Graphics Processing Units (GPU) has also played its role. The field of Computational Fluid Dynamics (CFD) has benefited from these developments with a large amount of effort focusing on the parallelization of common CFD algorithms.

With the increase in popularity of GPU and OpenMP parallel computing, an increasing number of researchers are investigating classical vector style parallelization using Single Instruction on Multiple Data set (SIMD) algorithms, which are particularly well suited to GPU computation [1]. Examples of such schemes are the family of Kinetic-Theory based solvers which employ vector splitting in their flux calculation [2,3] and are hence able to allow a large fraction of their programs to employ "embarrassingly parallel" flux and state computations [3]. Other vector split solvers [1,4], while not borne from Kinetic Theory, are also well suited to this form of parallel computation due to the high locality of the schemes. While these schemes are not without their flaws, the high degree of parallelization and scalability makes their use appealing to large scale engineering applications.

Modern CPU designs incorporate both scalar and vector processors. Good examples are the Cell processor, containing a single scalar processor and eight vector processors [5], and the modern Intel Sandy Bridge and Ivy Bridge CPUs. With the introduction of the Sandy Bridge core came the extension of the x86 instruction set known as the Advanced Vector eXtensions (AVX), first proposed by Intel in 2008 as the logical successor to the Streaming SIMD Extensions (SSE) used on earlier CPUs. The basic premise of the AVX concept is the ability of an AVX enabled core to perform a single instruction across a set of data contained within the AVX registers. These registers – 256 bits in length – allow for SIMD style parallel computation upon eight floating point variables (or 4 double variables). Previous research into the use of AVX to support CFD computation have shown that a large fraction of the performance increase between AVX and older SSE-based CPUs is due to the use of the AVX extensions [6].

With the introduction of the E5 series of Intel Xeon CPUs, which contain up to 8 physical cores and are designed to be employed in dual CPU systems (allowing a total of 16 physical cores), it seems only natural that the AVX registers present in each core be used to their maximum potential for application to CFD research. In order to fairly compare the performance of a GPU enabled system containing a dual-CPU configuration, or to know if the increase in performance of a hybrid CPU/GPU implementation is worth the cost of communication between the GPU and CPU, the optimal and complete performance of a dual CPU system must be evaluated. Here we demonstrate the application of a Kinetic-Theory based vector split solver for the three dimensional Euler Equations (QDS) [7], which has previously demonstrated significant accelerations through the use of GPU parallelization, using a hybrid OpenMP and AVX acceleration which (theoretically) makes full use of the available CPU resources of a system. This is done through the sole use (where possible) of AVX intrinsic functions as opposed to compiler based optimization techniques. A detailed description of the vector computation kernels employed in the CFD computation will be shown and a comparison of the assembly code for our hand-written kernels using AVX intrinsic functions and the compiler optimized kernels will also be discussed.

2. The Quiet Direct Simulation (QDS) Method

The Quiet Direct Simulation (QDS) method falls into the category of Kinetic Theory based solvers which aim to numerically solve for fluxes at cell interfaces through the approximation of moments around a governing particle distribution function. This flux computation has been performed both through the use of simulation particles [8] and through net fluxes [7], though the two approaches can be shown to be equivalent. In its simplest form, QDS can be shown to be an approximation of the Equilibrium Flux Method [9], where fluxes are computed as:

$$F^{+} = \int_{0}^{\infty} f(v_n) v_n Q dv_n \qquad F^{-} = \int_{-\infty}^{0} f(v_n) v_n Q dv_n \tag{1}$$

where $f(v_n)$ is the equilibrium velocity probability distribution function, given by:

$$f(v) = \frac{1}{\sqrt{2\pi}s} \exp\left[\frac{-(v-\bar{v})^2}{2s^2}\right] \tag{2}$$

where v is the particle velocity, s is the standard deviation of the distribution function ($s=RT^{1/2}$) and the bulk velocity is indicated by the overscore. The conserved quantities – carried by each particle – are given by the vector:

$$Q=\left\{\rho,\rho v_x,\rho v_y,\rho v_z,\rho\left(\frac{1}{2}\left(v_x^2+v_y^2+v_z^2\right)+E_{in}\right)\right\} \tag{3}$$

The resulting flux expressions can be found in [9] and will not be reviewed in detail here. It its simplest, direction decoupled [10] form, the computation of each split flux can be replaced by a truncated numerical integration through the use of Gauss-Hermite Quadrature. The computation of the forward split fluxes for QDS can be shown as:

$$F^{+}_{\eta,QDS}=\int_0^{\infty}(v'+u)\eta^{+}f(v')dv'\approx\sum_{j=1}^{N}H_S(v_j)v_jw_j\eta_j^{+} \tag{4}$$

Similarly, the reverse fluxes can be evaluated as:

$$F^{-}_{\eta,QDS}=\int_{-\infty}^{0}(v'+u)\eta^{-}f(v')dv'\approx\sum_{j=1}^{N}H_S(-v_j)v_jw_j\eta_j^{-} \tag{5}$$

where η_j are the conserved macroscopic properties of the jth QDS "velocity bin" given as:

$$\eta_j=\begin{bmatrix}\rho\\ \rho v_j\\ \rho\left(\frac{1}{2}v_j^2+\varepsilon_j\right)\end{bmatrix} \tag{6}$$

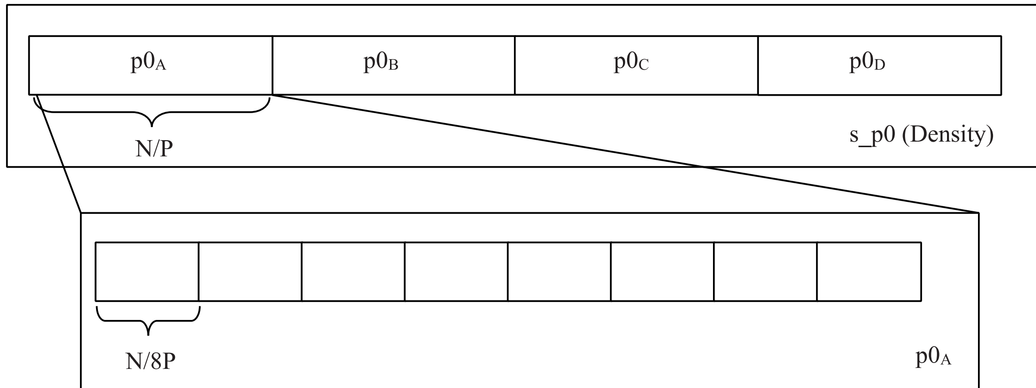
where $v_j=u+\sqrt{2\sigma^2}q_j$ is the velocity of the bin and ε_j is the internal energy of molecular structure in the bin such as rotational, vibrational, or electronic energy. Further details of the derivation – including its extension to higher order accuracy – will not be covered here. The core of the first order QDS method can be broken down into three distinct parts:

1. The split flux calculation: In each cell, the forward and backward fluxes are computed based on local conditions. This phase is "embarrassingly parallel" and can be safely described as a vector computation. Each thread is required to only perform computations relevant to its own cell, hence coalesced memory access is feasible.
2. The net flux calculation: In each cell, the net fluxes on either side of each interface associated with it are computed and used to update the state (i.e. the value of conserved quantities) in that cell. While also classified as an SIMD process, guaranteeing coalesced memory access is more of a challenge.
3. The primitive calculation: Not absolutely required, but very convenient, the value of primitives (i.e. density, velocity, and temperature) are computed. Much like the split flux calculation, this phase is easily vector parallelized.

Advanced implementations of the QDS method will require additional kernels for the computation of flux gradients, shear and heat stress gradients (for application to the Navier-Stokes equation). Previous implementations of QDS applied to GPU computation has demonstrated a speedup – defined as the ratio of the computational time required by the GPU and a single, optimized CPU core – of approximately 75x using an Nvidia M2070 (Fermi-based) computing GPU device. The optimization of a single core allowed a limited amount of AVX use, however, as we will show in later sections, the use of compilers in implementing AVX is still lacking.

3. Hybrid OpenMP – AVX parallelization

The application of OpenMP and AVX for parallelization of the QDS algorithm is relatively straightforward. A one dimensional data structure containing N elements holding our problem variables is shown in Figure 1. Each OpenMP thread will manage (*N/P*) elements – where *P* is the number of threads – is further subdivided into groups of 8 floating point variables, giving *N_AVX* = (*0.125N/P*) groups in total. Looping over these elements using AVX for the computation of the conserved quantities U from primitive values P is also shown in Figure 1, together with its conventional (un-optimized) equivalent. The key to high performance is the exclusive use (where possible) of AVX intrinsic functions and their associated variables within the computation kernel. Each variable employed in AVX_STATE_P2U contains floating point variables representing 8 values from the physical flow solver. Uncoalesced memory access for the state and flux function (not shown) is not problematic due to the high locality of the solver. Hence, retrieving information from outside the AVX or OpenMP boundary is also not a concern. For the computation of the new state (following computation of fluxes), a hybrid approach is employed with loops over N/P elements (for each OpenMP) containing sub-loops for optimized AVX use.




```
void SSE_STATE_P2U(__m256 *p0, __m256 *p1, __m256 *p2, __m256 *p3, __m256 *p4,
                   __m256 *u0, __m256 *u1, __m256 *u2, __m256 *u3, __m256 *u4,
                   float R, float GAMMA, int N_SSE) {
    __m256 v2, Temp1, Temp2;
    __m256 Half = _mm256_set1_ps(0.5f);  __m256 CV = _mm256_set1_ps(R/(GAMMA-1.0));
    for (int i = 0; i < N_SSE; i++) {
        // Compute conserved quantities from primitive quantities
        u0[i] = p0[i];
        u1[i] = _mm256_mul_ps(p0[i], p1[i]);
        u2[i] = _mm256_mul_ps(p0[i], p2[i]);
        u3[i] = _mm256_mul_ps(p0[i], p3[i]);
        Temp1 = _mm256_mul_ps(p1[i], p1[i]);      Temp2 = _mm256_mul_ps(p2[i], p2[i]);
        v2 = _mm256_add_ps(Temp1, Temp2);         Temp1 = _mm256_mul_ps(p3[i], p3[i]);
        v2 = _mm256_add_ps(v2, Temp1);
        v2 = _mm256_mul_ps(v2, Half);             Temp1 = _mm256_mul_ps(CV, p4[i]);
        Temp1 = _mm256_add_ps(Temp1, v2);         u4[i] = _mm256_mul_ps(p0[i], Temp1);
    }
}
```

```
void Calc_State(float *p0, float *p1, float *p2, float *p3, float *p4,
                float *u0, float *u1, float *u2, float *u3, float *u4,
                float R, float GAMMA, int N) {
    float CV = (R/(GAMMA-1.0));
    for (int i = 0; i < N; i++) {
        u0[i] = p0[i];            u1[i] = p0[i]*p1[i];
        u2[i] = p0[i]*p2[i];      u3[i] = p0[i]*p3[i];
        u4[i] = p0[i]*( CV*p4[i] + 0.5*(p1[i]*p1[i] + p2[i]*p2[i] + p3[i]*p3[i]));
    }
}
```

Fig. 1. [Top] Simple 1D representation of data structure used for hybrid OpenMP/AVX implementation. [Bottom] Comparison of state function using conventional (unoptimized) code (Calc_State) and it's AVX equivalent (AVX_STATE_P2U)

```
/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(…) shared (…) {
        /* Obtain thread number */
        tid = omp_get_thread_num();
        // Set the index for access to different parts of memory
        index = tid*N_SSE;
        p0 = (__m256*) s_p0 + index;        // Assign desired value (i.e. density)
        …
        Fp0 = (__m256*) s_Fp0 + index;      // Assign forward / backward fluxes
        …
        i = 0;
        while (i < CYCLES) {

                SSE_FLUX(…);
                SSE_TIMESTEP(…);
                SSE_STATE_U2P(…);
                #pragma omp barrier
                i++;
        }
  }  /* All threads join master thread and disband */
```

Fig. 2. Incomplete (semi pseudo code) of main function showing OpenMP implementation and the mapping of AVX registers.

```
771:    c5 f0 14 c9             vunpcklps %xmm1,%xmm1,%xmm1
 775:   c5 f8 5a c9             vcvtps2pd %xmm1,%xmm1
 779:   c5 f3 59 ce             vmulsd %xmm6,%xmm1,%xmm1
 77d:   c5 d3 58 c9             vaddsd %xmm1,%xmm5,%xmm1
 781:   c5 db 59 c9             vmulsd %xmm1,%xmm4,%xmm1
 785:   c5 fb 12 c9             vmovddup %xmm1,%xmm1
 789:   c5 f9 5a c9             vcvtpd2ps %xmm1,%xmm1

 a1f:   c5 fc 5e 04 07          vdivps (%rdi,%rax,1),%ymm0,%ymm0
 a24:   c5 fc 29 04 02          vmovaps %ymm0,(%rdx,%rax,1)
 a29:   c4 c1 7c 28 04 04       vmovaps (%r12,%rax,1),%ymm0
 a2f:   c5 fc 5e 04 07          vdivps (%rdi,%rax,1),%ymm0,%ymm0
 a34:   c5 fc 29 04 01          vmovaps %ymm0,(%rcx,%rax,1)
 a39:   c5 fc 59 c0             vmulps %ymm0,%ymm0,%ymm0
 a3d:   c5 fc 28 0c 06          vmovaps (%rsi,%rax,1),%ymm1
 a42:   c5 f4 59 d9             vmulps %ymm1,%ymm1,%ymm3
```

Fig. 3. Incomplete listing of assembly code for [Top] the conventional state function using –O3 optimization, and [Bottom] the OpenMP/AVX state function also using –O3 optimization.

4. Performance

The assembly code produced following compilation was produced using g++ (Version 4.4.3, 64 bit build) and is shown in Figure 3. Each of the codes compiled were contained within the same file and compiled simultaneously using –O3 optimization. The computation was performed on a modest problem set (80^3 ~ 0.51million cells) for 1000 time steps. The assembly code shown indicates that the g++ compiler has only employed SSE intrinsic functions in its optimization (note the *xmm* registers used). However, the g++ compiler has successfully compiled the AVX intrinsic functions – as noted by the use of the *ymm* registers in the assembly code in Figure 3. It is therefore worth noting that, despite the high degree of optimization requested during compilation, AVX instructions will not be used with g++ unless specifically called using intrinsic functions. It is also worth noting that the assembly code for the AVX state function is smaller than the conventional state function, and that a larger fraction of the code employs AVX related computations. When using a dual-CPU

configuration using 2x E52670 Xeon CPU's (16 physical cores total) and AVX, a computational speedup of 177 times was reported against using a single core using –O3 (i.e. SSE) optimization for both of the flux and state computation procedures. This is indeed super-linear performance, since the theoretical maximum should be 128 times – the remaining speedup may be attributed to cache effects.

5. Conclusion

The straightforward implementation of a hybrid OpenMP / AVX implementation for a kinetic-theory based vector split Finite Volume Method (QDS) has been demonstrated and tested. The majority of the QDS method, much like the remaining group of Boltzmann – based solvers, has a high degree of locality and is easy accelerated using vector processors and SIMD-based parallelization methods. The derivation of the QDS fluxes in direction decoupled form has been presented. One of these kernels (the state computation kernel) and a simplified reconstruction of the main() function have been detailed and discussed. Through the design of computation kernels based exclusively on the use of AVX intrinsic functions and their associated data types, a speedup of 177x is reported against a single core using SSE vector processes. Assembly code created from kernels based heavily (or solely, where possible) on AVX intrinsic functions also demonstrate smaller code size. This investigation demonstrates that a dual-core system (tested using a pair of E52670 Xeon CPU's), when using code specially prepared to take advantage of all available cores and full AVX acceleration, is capable of matching (or beating) the performance previously reported for GPU accelerated applications.

Acknowledgements

The authors acknowledge financial support from the Taiwan National Science Council (Grant Number NSC 99-2221-E-492 -005 -MY3). We are also grateful to Acer, Nvidia and Intel for various loaned and donated hardware components.

References

[1] F.-A. Kuo, M.R. Smith, C.-W. Hsieh, C.-Y. Chou and J.-S. Wu, GPU acceleration for general conservation equations and its application to several engineering problems, Computers and Fluids, 45[1]: pp. 147-154, 2011.
[2] Lin L. S., Chang, H. W. and Lin, C. A., 2012, Multi relaxation time lattice Boltzmann simulations of transition in deep 2D lid driven cavity using GPU. Computers and Fluids, doi:10.1016/j.compfluid.2012.01.018
[3] M.R. Smith F.-A. Kuo, C.-Y. Chou, J.-S. Wu and H.M. Cave, Application of a Kinetic Theory based solver of the Euler Equations using GPU, International Conference on Parallel Computational Fluid Dynamics 2009, NASA Ames (San Francisco), May 2009.
[4] Van Leer B. Towards the ultimate conservative difference scheme III. Upstream-centered finite-difference schemes for ideal compressible flow. Journal of Computational Physics. 1977;23:263-75.
[5] M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe and T. Yamazaki, Synergistic Processing in Cell's Multicore Architecture, IEEE Micro, 2006: 26[2], pp: 10-24.
[6] A. Vladimirov, Arithmetics on Intel's Sandy Bridge and Westmere CPUs: not all FLOPs are created equal, in Colfax Research Report, March 2012, http://research.colfaxinternational.com/post/2012/04/30/FLOPS.aspx (Last Viewed 20th March, 2013).
[7] Smith MR, Cave H, Wu JS, Jermy MC, Chen YS. An improved quiet direct simulation method for Eulerian fluids using a second-order scheme. Journal of Computational Physics. 2009;228:2213-24.
[8] Albright BJ, Daughton W, Lemons DS, Winske D, Jones ME. Quiet direct simulation of plasmas. Physics of Plasmas. 2002;9:1898-904.
[9] Pullin DI. Direct simulation methods for compressible inviscid ideal-gas flow. Journal of Computational Physics. 1980;34:231-44.
[10] Smith MR, Macrossan MN, Abdel-jawad MM. Effects of direction decoupling in flux calculation in finite volume solvers. Journal of Computational Physics. 2008;227:4142-61.