PARALLELIZATION OF NUMERICAL METHODS ON PARALLEL PROCESSOR ARCHITECTURES

Endre László

Theses of the Ph.D. Dissertation

Pázmány Péter Catholic University
Faculty of Information Technology and Bionics
Roska Tamás Doctoral School of Sciences and Technology

Thesis Advisor:
Prof. Dr. Péter Szolgay, D.Sc.

Budapest, 2015

1 Introduction

Today the development of every scientific, engineering and financial field that rely on computational methods is severely limited by the stagnation of computational performance caused by the physical limits in the VLSI (Very Large Scale Integration) technology. This is caused by the single processor performance of the CPU (Central Processing Unit) due to vast heat dissipation that can not be increased. Around 2004 this physical limit made it necessary to apply more processors onto a silicon die and so the problem of efficient parallelization of numerical methods and algorithms on a processor with multiple cores was born.

The primary aim of my thesis work is to take advantage of the computational power of various modern parallel processor architectures to accelerate the computational speed of some of the scientific, engineering and financial problems by giving new algorithms and solutions.

Limits of physics Gordon Moore in the 1960's studied the development of the CMOS (Complementary Metal-Oxide-Semiconductor) manufacturing process of integrated circuits (IC). In 1965 he concluded from five manufactured ICs, that the number of transistors on a given silicon area will double every year. This finding is still valid after half a century with a major modification: the number of transistors on a chip doubles every two years (as opposed to the original year).

This law is especially interesting since around 2004 the manufacturing process reached a technological limit, which prevents us from increasing the clock rate of the digital circuitry. This limit is due to the heat dissipation. The amount of heat dissipated on the surface of a silicon die of size of a few square centimetres is in the order 100s of Watts. This is the absolute upper limit of the TDP (Thermal Design Power) that a processor can have. The ever shrinking VLSI (Very Large Scale Integration) feature sizes cause: 1) the resistance (and impedance) of conductors to increase; 2) the parasitic capacitance to increase; 3) leakage current on insulators like the MOS transistor gate oxide to increase. These increasing resistance and capacitance related parameters limit the clock rate of the circuitry. In order to increase the clock rate with these parameters the current needs to be increased and that leads to increased power dissipation. Besides limiting the clock rate the data transfer rate is also limited by the same physical facts.

The TDP of a digital circuit is composed in the following way: $P_{TDP} = P_{DYN} + P_{SC} + P_{LEAK}$, where P_{DYN} is the dynamic switching, P_{SC} is the instantaneous short circuit (due to non-zero rise or fall times) and P_{LEAK} is the leakage current induced power dissipation. The most dominant power dissipation is due to the dynamic switching $P_{DYN} \propto CV^2 f$, where $\propto$ indicates proportionality, f is the clock rate, C is the capacitance arising in the circuitry and V is the voltage applied to the capacitances.

Parallelisation - temporary solution to avoid the limits of physics Earlier, the continuous development of computer engineering meant increasing the clock rate and the number of transistors on a silicon die, as it directly increased the computational power. Today, the strict focus is on increasing the computational capacity with new solutions rather than increasing the clock rate on the chip. One solution is to parallelise on all levels of a processor architecture with the cost of cramming more processors and transistors on a single die. The parallelisation and specialization of hardware necessarily increases the hardware, algorithmic and software complexity. Therefore, since 2004 new processor architectures appeared with multiple processor cores on a single silicon die. Also, more and more emphasise is put to increase the parallelism on the lowest levels of the architecture. As there are many levels of parallelism built into the systems today, the classification of these levels is important to understand what algorithmic features can be exploited during the development of an algorithm.

Amdahl's law Gene Amdahl at a conference in 1967 [1] gave a talk on the achievable scaling of single processor performance to multiple processors assuming a fixed amount of work to be performed. Later, this has been formulated as Eq. (1) and now it is know as Amdahl's law. Amdahl's law states that the speedup due to putting the P part of the workload of a single processor onto N identical processors results in $S(N)$ speedup compared to single processor.

$$S(N) = \frac{1}{(1-P) + \frac{P}{N}} \tag{1}$$

One may think of the proportion P as the workload that has been parallelized by an algorithm (and implementation). The larger this proportion, the better the scaling of the implementation will be. This setup is also known as strong scaling. This is a highly important concept that is implicitly used in the arguments when parallelization is discussed.

Gustafson's law John Gustafson in 1988 reevaluated Amdahl's law [2] and stated that in the framework of weak scaling the speedup is given by Eq. (2). Weak scaling measures the execution time when the problem size is increased N fold when the problem is scheduled onto N parallel processors. Eq. (2) states that the execution time ($S_{latency}(s)$) decreases when the latency due to the P proportion - which benefits from the parallelization - speeds up by a factor of s.

$$S_{latency}(s) = 1 - P + sP \tag{2}$$

2 Classification of Parallelism

The graph presented on Figure 1. shows the logical classification of parallelism on all levels. Leaves of the graph show the processor features and software components that implement the parallelism. Processor and software features highlighted as GPU (Graphics Processing Unit) and CPU/MIC (MIC - Many Integrated Core) features are key components used in the work.

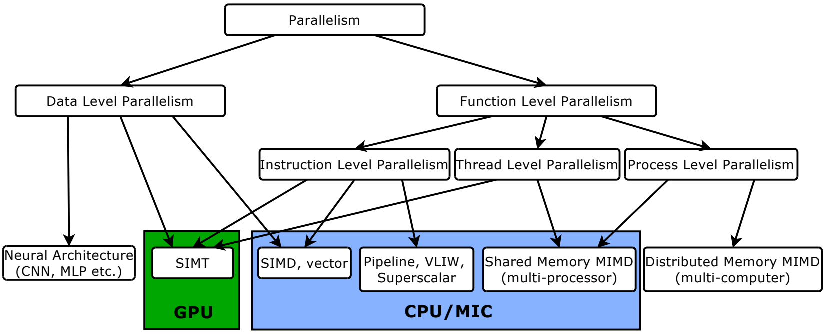
Parallelism
Data Level Parallelism
Function Level Parallelism
Instruction Level Parallelism
Thread Level Parallelism
Process Level Parallelism
Neural Architecture (CNN, MLP etc.)
SIMT
GPU
SIMD, vector
Pipeline, VLIW, Superscalar
Shared Memory MIMD (multi-processor)
CPU/MIC
Distributed Memory MIMD (multi-computer)

Figure 1: Classification of parallelism along with parallel computer and processor architectures that implement them.

Architecture dependent performance bounds Depending on the design aims of a parallel processor architecture the computational performance of these architectures vary with the problem. The performance can be bound by the lack of some resources needed for that particular problem, eg. more floating point units, greater memory bandwidth etc.

Therefore it is common to refer to an implementation being: 1) *compute bound* if further performance gain would only be possible with more compute capability or 2) *memory bandwidth bound* if the problem would gain performance from higher memory bandwidth. The roofline model [3] helps identifying whether the computational performance of the processor on a given algorithm or implementation is bounded by the available computational or memory controller resources and also helps approximating the extent of the utilization of a certain architecture. See Figure 2 for comparing parallel processor architectures used in the dissertation. Processor architectures that were recently introduced to the marker and which are to be announced are also noted on the figure.

On Figure 2 the graphs were constructed based on the maximum theoretical computational capacity (GFLOP/s) and data bandwidth (GB/s) metrics. In the case of FPGAs the number of implementable multipliers and the number of implementable memory interfaces working at the maximum clock rate gives the base for the calculation of the graphs. The exact processor architectures behind the labels are the following: 1) GPU-K40: NVIDIA Tesla K40m card with Kepler GK110B microarchitecture working with “Boost” clock rate; 2) GPU-K80: NVIDIA Tesla K80 card with Kepler GK210 microarchitecture working with “Boost” clock rate; 3) CPU-SB: Intel Xeon E5-2680 CPU with Sandy Bridge microarchitecture; 4) CPU-HW: Intel Xeon E5-2699v3 CPU with Haswell microarchitecture; 5) MIC-KNC: Intel Xeon Phi 5110P co-processor card with Knights Corner microarchitecture; 6) MIC-KNL: Intel Xeon Phi co-processor with Knights Landing microarchitecture - the exact product signature is yet to be announced; 7) FPGA-V7: Xilinx Virtex-7 XC7VX690T; 8) FPGA-VUSP: Xilinx Virtex UltraScale+ VU13P.

As the aim of my work was achieving the fastest execution time by parallelization the roofline model is used to study the computational performance of the selected problems where applicable.

Parallel Processor Architectures and Languages A vast variety of parallel architectures are used and experimented in HPC (High Performance Computing) to compute scientific problems. Multi-core Xeon class server CPU by Intel is the leading architecture used nowadays in HPC. GPUs originally used for graphics-only computing has become a widely used architecture to solve certain problems. In recent years Intel introduced the MIC (Knights Corner microarchitecture) architecture in the Xeon Phi coprocessor family. IBM with the POWER and Fujitsu with the SPARC processor families are focusing on HPC. FPGAs (Field

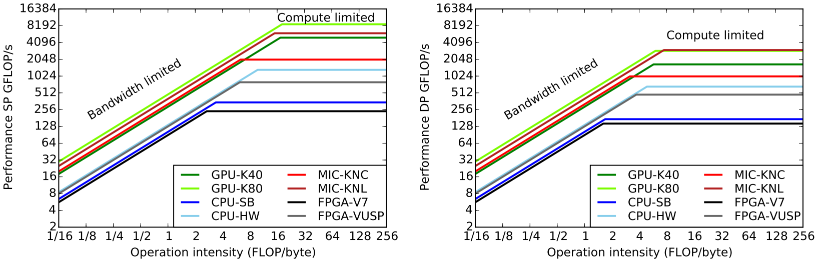

Figure 2: Roofline model for comparing parallel processor architectures. Note: SP stands for Single Precision and DP stands for Double Precision.

Programmable Gate Array) by Xilinx and Altera are also used for the solution of some special problems. ARM as an IP (Intellectual Property) provider is also making new designs for x86 CPU processors which find application in certain areas of HPC. Also, research is conducted to create new heterogenous computing architectures to solve the power efficiency and programmability issues of CPUs and accelerators.

Programming languages like FORTRAN, C/C++ or Python are no longer enough to exploit the parallelism of these multi- and many-core architectures in a productive way. Therefore, new languages, language extensions, libraries, frameworks and DSLs (Domain Specific Language) appeared in recent years, see Figure 3. Without completeness the most important of these are: 1) CUDA (Compute Unified Device Architecture) C for programming NVIDIA GPUs; 2) OpenMP (Open Multi Processing) directive based languages extension for programming multi-core CPU or many-core MIC architectures; 3) OpenCL (Open Compute Language) for code portable, highly parallel abstraction; 4) AVX (Advanced Vector eXtension) and IMCI (Initial Many Core Instruction) vectorized, SIMD (Single Instruction Multiple Data) ISA (Instruction Set Architecture) and intrinsic instructions for increased ILP (Instruction Level Parallelism) in CPU and MIC; 5) OpenACC (Open Accelerators) directive based language extension for accelerator architectures; 6) HLS (High Level Synthesis) by Xilinx for improved code productivity on FPGAs.

All these new architectural and programming features raise new ways to solve existing parallelisation problems, but non of them provide high development productivity, code- and performance portability as one solution. Therefore, these problems are the topic of many ongoing research in the HPC community.

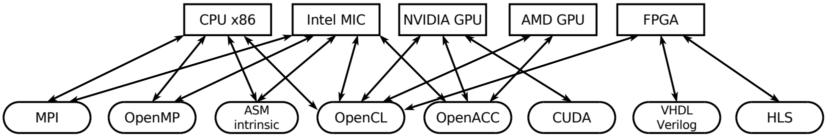

Figure 3: Relations between processor architectures, languages and language extensions.

Classification of scientific problems according to parallelization Classification of the selected problems is based on the 13 “dwarves” of “A view of the parallel computing landscape” [4]. My results are related to the dwarves highlighted with bold fonts:

1. Dense Linear Algebra
2. **Sparse Linear Algebra**
3. Spectral Methods
4. N-Body Methods
5. **Structured Grids**
6. **Unstructured Grids**
7. MapReduce
8. Combinational Logic
9. Graph Traversal
10. Dynamic Programming
11. Backtrack and Branch-and-Bound
12. Graphical Models
13. Finite State Machines

3 Selected and Parallelized Numerical Problems

The selected numerical algorithms cover the fields of numerical mathematics that aim for the solution of PDEs (Partial Differential Equation) in different engineering application areas, such as CFD (Computational Fluid Dynamics), financial engineering, electromagnetic simulation or image processing.

3.1 Tridiagonal System of Equations

Engineering, scientific and financial applications often require the simultaneous solution of a large number of independent tridiagonal systems of equations with varying coefficients [5, 6, 7, 8, 9, 10]. The solution of tridiagonal systems also arises when using line-implicit smoothers as part of a multi-grid solver [11], and when using high-order compact differencing [12, 13]. Since the number of systems is large enough to offer considerable parallelism on many-core systems, the choice between different tridiagonal solution algorithms, such as Thomas, CR (Cyclic Reduction) or PCR (Parallel Cyclic Reduction) needs to be re-examined. In my work I developed and implemented near optimal scalar and block tridiagonal algorithms for CPU, Intel MIC and NVIDIA GPU with a focus on minimizing the amount of data transfer to and from the main memory using novel algorithms and register blocking mechanism, and maximizing the achieved bandwidth. The latter means that the achieved computational performance is also maximized (see Figure 4) as the solution of tridiagonal system of equations is bandwidth limited due to the operation intensity.

On Figure 4 the computational upper limits on the graphs were constructed based on the maximum theoretical computational capacity (GFLOP/s) and data bandwidth (GB/s) metrics of the three processors. The operation intensity is calculated from the amount of data that is moved through the memory bus and the amount of floating point operations performed on this data. The total amount of floating point operations is calculated and divided by the execution time to get the performance in GFLOP/s. The operation intensity and GFLOP/s performance metrics are calculated for each solver and they are represented with stars on the figures.

In most cases the tridiagonal systems are scalar, with one unknown per grid point, but this is not always the case. For example, computational fluid dynamics applications often have systems with block-tridiagonal structure up to 8 unknowns per grid point [7]. The solution of block tridiagonal system of equations are also considered which are sometimes required in CFD (Computational Fluid Dynamic) applications. A novel work-sharing and register blocking based Thomas solver for GPUs is also created.

3.2 Alternating Directions Implicit Method

The numerical approximation of multi-dimensional PDE problems on regular grids often requires the solution of multiple tridiagonal systems of equations. In engineering applications and computational finance such problems arise frequently as part of the ADI (Alternating Direction Implicit) time discretization favored by many in the community, see [10]. The ADI method requires the solution of multiple tridiagonal systems

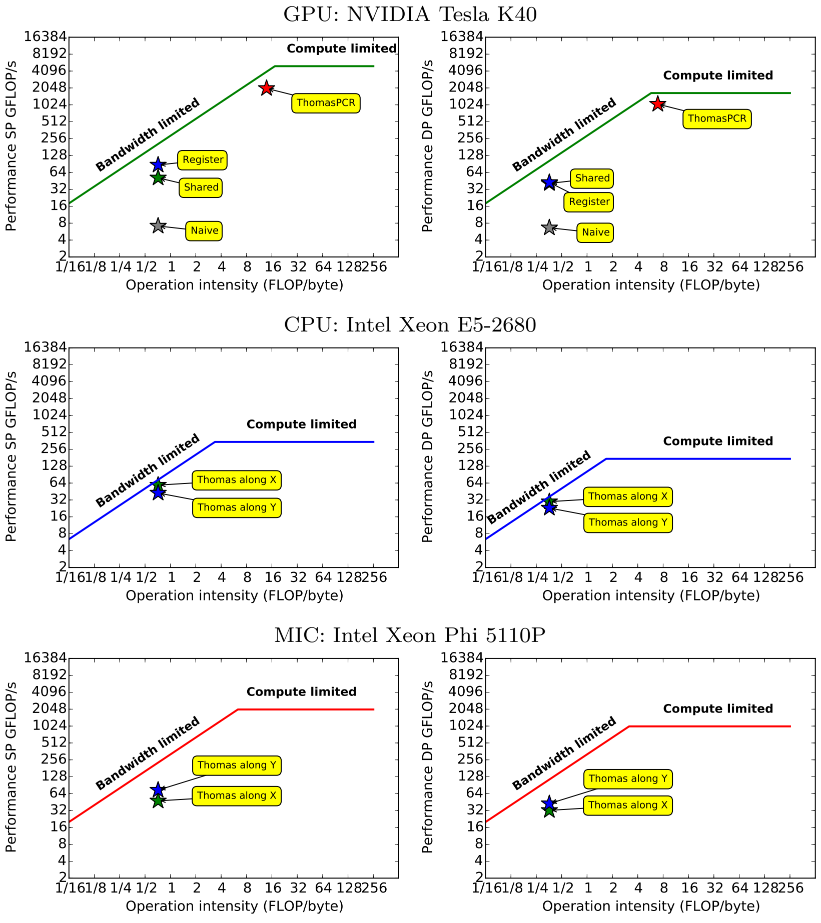

Figure 4: Roofline model applied to the implemented scalar tridiagonal solvers on GPU, CPU and MIC processor architectures. The proximity of stars to the upper computational limits shows the optimality of the implementation on the architecture.

of equations in each dimension of a multi-dimensional problem, see [14, 9, 15, 16].

3.3 Cellular Neural Network

The CNN (Cellular Neural Network) [17] is a powerful image processing architecture whose hardware implementation is extremely fast [18, 19]. The lack of such hardware device in a development process can be substituted by using an efficient simulator implementation. A GPU based implementation of a CNN simulator using NVIDIA's Fermi architecture provides a good alternative. Different implementation approaches are considered and compared to a multi-core, multi-threaded CPU and some earlier GPU implementations. A detailed analysis of the introduced GPU implementation is presented.

3.4 Computational Fluid Dynamics

Achieving optimal performance on the latest multi-core and many-core architectures depends more and more on making efficient use of the hardware's vector processing capabilities. While auto-vectorizing compilers do not require the use of vector processing constructs, they are only effective on a few classes of applications with regular memory access and computational patterns. Other application classes require the use of parallel programming models, and while CUDA and OpenCL are well established for programming GPUs, it is not obvious what model to use to exploit vector units on architectures such as CPUs or the MIC. Therefore it is of growing interest to understand how well the Single Instruction Multiple Threads (SIMT) programming model performs on a an architecture primarily designed with Single Instruction Multiple Data (SIMD) ILP parallelism in head. In many applications the OpenCL SIMT model does not map efficiently to CPU vector units. In my dissertation I give solutions to achieve vectorization and avoid synchronization - where possible - using OpenCL on real-world CFD simulations and improve the block coloring in higher level parallelization using matrix bandwidth minization reordering.

4 New scientific results

The new scientific results are grouped into thesis groups according to their classification among the 13 dwarves. Results regarding the new solutions proposed for solving tridiagonal system of equations can be categorized as the "Sparse Linear Algebra" dwarf which are detailed in Thesis group I. Image processing and PDE solution using the ADI method is categorized as the "Structured Grid" dwarf which is detailed in Thesis group II. Finally, results regarding CFD computations are categorized as the "Unstructured Grid" dwarf which is detailed in Thesis group III. The relations between thesis groups, parallel problem classification and parallel processor architectures are summarized in Figure 5.

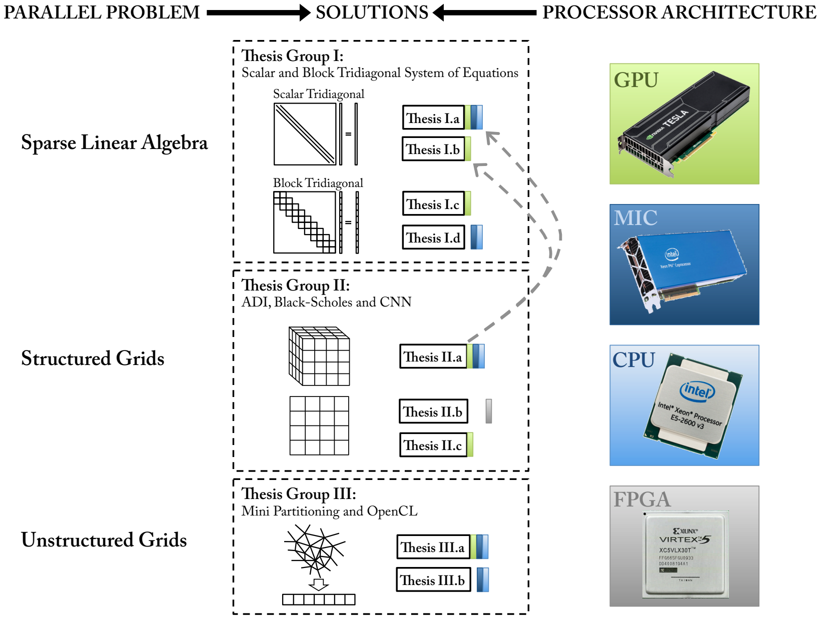

Figure 5: Thesis groups: relation between thesis groups, parallel problem classification and parallel processor architectures. Colored vertical bars right to thesis number denote the processor architecture that is used in that thesis. Dashed grey arrow represent relation between theses.

New scientific results are published in journals(marked as [J]), conference (marked as [C]) proceedings and conference talks (marked as [CT]).

Publications corresponding to the thesis groups are noted below.

Thesis Group I. Efficient algorithms for sparse linear algebra

Many times PDEs arising in the scientific, engineering and financial applications require an extensive use of sparse linear algebra which needs to be efficiently parallelised for current and upcoming parallel processor architectures. In particular, the numerical solution of some special parabolic, diffusion type PDEs with implicit solvers boil down to the solution of tridiagonal system of equations where the elements of the tridiagonal matrix are either scalar values or blocks with size $M \times M$, where $M \in [2, 8]$. New parallel algorithms for the acceleration of such tridiagonal solvers is therefore essential to these scientific, engineering and financial communities to accelerate research and innovation. Theses in this group contribute to the parallelisation and acceleration of such methods on CPU, GPU and MIC architectures.

Corresponding publications: [J1], [C1], [CT1], [C2], [C3]

Thesis I.a *I have developed new register blocking based local data transposition algorithms for multi- and many-core parallel processor architectures to improve the memory access pattern of the Thomas algorithm when solving tridiagonal system of equations where the coefficients are stored in consecutive order in the memory. The overall performance gain is: 1) up to* $\times 4.3$ *on the GPU compared to the NVIDIA cuSPRASE library; 2) up to* $\times 1.5$ *on the CPU compared to the Intel MKL library and 3) up to* $\times 1.9$ *on the MIC compared to the Intel MKL library.*

A tridiagonal system of equation is composed of three coeffcient vectors **a**, **b** and **c**, one unknown vector **u** and the right hand side vector **d**. All these vectors are element of $\mathbb{R}^N$ and they have identical data layout in the memory. The data layout of the coefficients of a tridiagonal system of equations may depend on the grid and the numerical method where it is applied, this can be consecutive or stride-N. Consecutive or stride-1 means that elements a_k and $a_{(k+1)}$ of vector **a** are at consecutive memory addresses in the linear memory, ie. at address k and $(k+1)$. Stride-N means that elements a_k and $a_{(k+1)}$ of vector **a** are at address $k \times N$ and $(k + 1) \times N$ in the linear memory. In many cases the consecutive data layout is used which results in poor cache-line utilization and therefore requires memory access pattern optimization. I have developed two algorithms for GPUs, to perform data load (and store) of multiple values at once and transpose these data for calculation. I have also developed similar transposition based solvers for the CPU and MIC architectures. My solutions allow the transposition to be done using registers and the use of data directly from registers.

Thesis I.b *I have developed an efficient implementation of a new hybrid algorithm for the solution of tridiagonal system of equations on GPU architectures using the conventional Thomas and PCR algorithms in the case when the varying coefficients are stored in the main memory of the GPU. This GPU specific solution allows the utilization of the large number of registers on GPU architectures and local transposition of data when reading the coefficients are done in registers. The resulting solver is up to* ×9 *times faster than the solver in the NVIDIA cuSPARSE library and up to* ×2.1 *times faster than the transposition based GPU solver introduced in Thesis I.a.*

I have created the implementation of a new hybrid algorithm which is a combination of the Thomas and the PCR algorithms, which is optimal in the sense that is limited either by the memory bandwidth or the compute capacity of the GPU, depending on the floating point precision. This hybrid GPU specific algorithm allows the utilization of registers which leads to extreme efficiency when the system size is sufficiently small to fit into the registers of the GPU. Consecutive or stride-N data layout can both be handled with this algorithm. When needed transposition in register can be performed without the use of shared memory. As there is no need for storing intermediate values in global or local memory as in the case of the regular Thomas algorithm the ratio of floating point operations per byte is much higher for the new hybrid algorithm which results in better performance on a high compute intensity architecture like the GPU.

Thesis I.c *I have developed a new algorithm for the solution of block tridiagonal system of equations on GPU using a new thread work sharing and register blocking approach, when the block sizes are* $M \times M$ *with* $M = \in 2, 8]$*. The achieved computational performance of this GPU specific work sharing approach is superior compared to the know algorithms and their implementation. I have experimentally showed that it is up to* ×27 *times faster than the Intel MKL banded solver and up to* ×9.8 *faster* *than the PCR-based GPU solver proposed by [20].*

Block tridiagonal system of equations arise in CFD (Computational Fluid Dynamic) applications [8, 7], where block sizes vary according to the multi-variable PDE. I gave a new thread work-sharing algorithm to parallelise the Thomas algorithm. I have also shown that this approach is computationally more efficient than the CR or PCR algorithms and that sufficient parallelism can be exploited with work-sharing to exceed the performance of the CR and PCR algorithms when block size is in the range $M =\in 2, 8]$.

Thesis I.d *I have developed a new implementation for the solution of block tridiagonal system of equations on multi-core CPU and many-core MIC with vector instructions, which outperforms the banded solver of the Intel MKL library in terms of computational performance by a factor of* $\times 6.5$ *and* $\times 5$ *in the case of the CPU and MIC repectively.*

I have restructured the code and data of the standard block Thomas algorithm with C++ templates and code transformation to achieve better data locality and guide the compiler in the vectorization procedure. The result is a highly efficient SIMD based CPU and MIC implementation of the block Thomas algorithm.

Thesis Group II. Efficient algorithms for strucutred grid computations

The solution of parabolic, diffusion type PDEs on a structured grid domain can be efficiently solved using the ADI (Alternating Direction Implicit) method which – in higher dimensions – boils down to the solution of multiple tridiagonal system of equations. The memory access pattern of the tridiagonal solves along different dimensions varies and results in significant performance loss if conventional multi- and many core implementations of numerical library functions are used. Therefore new efficient algorithms are needed. Also, the solution of diffusion type PDEs like the Black-Scholes PDE arising in financial applications and other PDEs related to CNN based image processing require efficient parallelization solutions for parallel processor architectures like FPGAs and GPUs.

Corresponding publications: [J1], [CT1], [C3], [C6]

Thesis II.a *I have elaborated and implemented new parallel algorithms to accelerate the calculation of the ADI (Alternating Direction Implicit) method for the solution of PDEs in higher spatial dimensions on CPU, MIC and GPU. The resulting ADI solver utilizes new solvers of tridiagonal system of equations with stride-1 and stride-N access patterns.*

When solving parabolic, diffusion type multi-dimensional PDEs many times it is possible to decouple the solution of the higher spatial N dimensions into N number of one-dimensional problems. I have developed ways to optimize the efficiency of the solution in each dimension as the memory access pattern changes significantly along each dimension for arbitrary dimensions. This result is also a representative use case for the tridiagonal solvers of Thesis I.a and Thesis I.b.

Thesis II.b *I have developed a power-efficient parallel FPGA based solver in HLS (High Level Synthesis) to accelerate the numerical solution of the 1-factor Black-Scholes PDE. The resulting FPGA solver is on par with the CPU solver in terms of computational performance, but it outperforms the CPU in terms of computational efficiency (GFLOPS/W) by a factor of 4 in the case of the explicit solution and by a factor of 1.3 when solving the implicit problem.*

Solving the 1-factor Black-Scholes PDE for pricing financial derivatives with one underlying asset requires an explicit or implicit solution of the PDE. I have proposed an efficient stencil operation type solutions for the explicit method and an efficient Thomas algorithm implementation for the implicit method.

Thesis II.c *I have experimentally proven that the utilization of the texture cache by a double buffer approach is an efficient way to implement a GPU based accelerator for the solution of the CNN state equation as it increases the cache hit rate of the two dimensional texture cache due to spatial locality and reduces the number of integer operation involved during the memory index calculations through the built in functionality of the texture cache.*

When solving the CNN state equation to perform a diffusion operation on an image, a heat diffusion PDE is solved. The solution of such a PDE is memory bandwidth limited. As such, a tiling or cache blocking optimizations amortize the data transfer and allow good performance.

I have shown that this performance can be overcome by utilizing the built-in texture cache capability of the GPU. With this approach the solution of the two dimensional CNN state equation can take advantage of the significant cache reuse of the texture cache when fetching data. Also, significant integer operations and latency can be saved by utilizing the built-in coordinate calculation units of the GPU.

Thesis Group III. Efficient algorithms for unstructured grid computations

Increasing the efficiency of parallel computations on unstructured grid applications in the OP2 framework is of high importance to the scientific community. Improving the single node scalability of the indirect increment of values in the OP2 framework is of great importance for performance. One component of the scalability is the mini-partitioning. The current naive mini-partitioning solution can be improved by exploiting the use minimal matrix bandwidth reordering techniques. New methods for an OpenCL backend of the OP2 framework can improve the performance of unstructured grid applications on CPU and MIC, and provide code portability.

Corresponding publications: [C4], [J2], [C5]

Thesis III.a *I have experimentally proven that the efficiency of the parallel colored indirect incrementation in unstructured grid computations can be improved by extending the mini-partitioning in OP2 using the GPS (Gibbs-Poole-Stockmeyer) minimal bandwidth reordering algorithm on real-world CFD simulation problems. This approach dramatically decreases the number of block colors used in the parallel increment and therefore increases the number of blocks within a color which can be incremented in parallel. The reduction in the number of block colors is up to* ×37.5 *for real-world test cases and the speedup resulting from this improvement is* ×7.4*.*

In order to increase the efficiency of parallel incrementation on unstructured grids OP2 utilizes a two level coloring scheme to identify the set of elements and blocks which can be incremented in parallel. The number of colors determines the number of sequential steps during the incrementation process. In real world applications the number of block colors can be high due to excessive number of connections between the mini-partitions. I have proposed the use of the GPS bandwidth minimization algorithm to reorder meshes so that the naïve mini-partitioning creates blocks with less neighboring connections which helps the two level coloring algorithm to get better block coloring.

Thesis III.b *I have analysed and proposed new heuristic code transformation techniques to improve the vectorization capabilities of OpenCL on CPU and MIC architectures. The resulting OpenCL kernels within OP2 lend themselves to better parallelization properties on real world simulation codes. In case of the CPU the achieved performance is on par with the OpenMP and MPI parallelization. However, the OpenCL implementation is* ×1.5 *faster on the MIC compared to the MPI+OpenMP solution.*

I have exploited the capabilities of the Intel OpenCL implementation for CPU and MIC to increase their performance by implementing the OpenCL backend of the OP2 framework. The key to this improvement is the matching of multi-threading and SIMD vectorization features of the CPU (or MIC) to the SIMT type kernel abstraction of the OpenCL standard.

5 Application of the results

As noted in the Section 3 the problems selected for parallelization originate from the scientific, engineering and financial disciplines and therefore they have a direct application in these fields. The results concerning scalar and block tridiagonal solvers were presented at the GPU Technology Conference in San Jose in 2014, the largest GPU conference by today and on conferences and journal papers..

The integration of the results from the research of scalar tridiagonal solver to atmospheric simulations is an ongoing work by Eike Mueller and his collegues at the Department of Mathematical Sciences at the University of Bath, UK. In atmospheric modelling due to the structured meshing of the atmosphere the cell geometries are too distorted which results in numerical stability problems, also known as highly anisotropic problem. Line-smoother in the multigrid preconditioner requires the frequent inversion of a tridiagonal matrix and therefore the Thomas-PCR algorithm may have a high impact on the performance.

My results are used to improve the performance of a Navier-Stokes CFD solver called Turbostream. Turbostream is developed by Dr. Tobias Brandvik and Dr. Graham Pullan in the Whittle Laboratory at the University of Cambridge, UK. It is aimed at solving CFD problems arising in the design of turbomachinery where the numerical solution of the Navier-Stokes equations are done using a line-implicit or semi-implicit Runge-Kutta method on an unstructured grid with stretched regions. Along these stretched regions an (block tridiagonal) implicit solution is required for unconditional stability, as the grid cells are highly anisotropic.

Dr. Serge Guillas and his colleagues at the University College London, UK used the VolnaOP2 tsunami simulation code – written by István Reguly and me – to perform measurements for the papers [21] and [22]. This implementation make large scale tsunami simulations feasible which result in statistically relevant results.

The Author's Publications

[J1] **Endre Laszló**, Mike Giles, and Jeremy Appleyard. "Manycore algorithms for batch scalar and block tridiagonal solvers (Accepted)". In: *ACM Transactions on Mathematical Software (TOMS)* (2015).

[C1] **Endre Laszló**, Zoltán Nagy, Mike Giles, István Reguly, Jeremy Appleyard, and Péter Szolgay. "Analisys of Parallel Processor Architectures for the Solution of Tridiagonal System of Equations". In: *International Symposium on Circuits and Systems.* Lisbon, Protugal: IEEE Press, 24-27 May 2015.

[CT1] **Endre László** and Mike Giles. "Efficient Solution of Multiple Scalar and Block-Tridiagonal Equations". In: *GPU Technology Conference.* San Jose, CA: NVIDIA, 2014.

[C2] Mike Giles, **Endre László**, István Reguly, Jeremy Appleyard, and Julien Demouth. "GPU Implementation of Finite Difference Solvers". In: *Proceedings of the 7th Workshop on High Performance Computational Finance.* WHPCF '14. New Orleans, Louisiana: IEEE Press, 2014, pp. 1–8. ISBN: 978-1-4799-7027-8. DOI: 10.1109/WHPCF.2014.10. URL: http://dx.doi.org/10.1109/WHPCF.2014.10.

[C3] **Endre László**, Michael B Giles, Jeremy Appleyard, and Péter Szolgay. "Methods to utilize SIMT and SIMD instruction level parallelism in tridiagonal solvers". In: *Cellular Nanoscale Networks and their Applications (CNNA), 2014 14th International Workshop on.* IEEE. 2014, pp. 1–2.

[C6] **Endre László**, Péter Szolgay, and Zoltán Nagy. "Analysis of a GPU based CNN implementation". In: *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on.* IEEE. 2012, pp. 1–5.

[C4] István Z Reguly, **Endre László**, Gihan R Mudalige, and Mike B Giles. "Vectorizing Unstructured Mesh Computations for Manycore Architectures". In: *Proceedings of Programming Models and Applications on Multicores and Manycores.* ACM. 2014, p. 39.

[J2] I Z. Reguly, **Endre László**, Gihan R. Mudalige, and Mike B. Giles. "Vectorizing unstructured mesh computations for many-core architectures". In: *Concurrency and Computation: Practice and Experience* (2015). ISSN: 1532-0634. DOI: 10.1002/cpe.3621. URL: http://dx.doi.org/10.1002/cpe.3621.

[C5] Mike B Giles, Gihan R Mudalige, Carlo Bertolli, Paul HJ Kelly, **Endre László**, and I Reguly. "An analytical study of loop tiling for a large-scale unstructured mesh application". In: *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:* IEEE. 2012, pp. 477–482.

The Author's Other Publications

Publications not strictly related to the doctoral theses

[J3] Béla Szentpáli, Gábor Matyi, Péter Fürjes, **Endre László**, Gábor Battistig, István Bársony, Gergely Károlyi, and Tibor Berceli. "Thermopile-based THz antenna". In: *Microsystem technologies* 18.7-8 (2012), pp. 849–856.

[C7] Béla Szentpáli, Gábor Matyi, Péter Fürjes, **Endre László**, Gábor Battistig, István Bársony, Gergely Károlyi, and Tibor Berceli. "THz detection by modified thermopile". In: *SPIE Microtechnologies* (2011).

[C8] **Endre László**, Kálmán Tornai, Gergely Treplán, and János Levendovszky. "Novel load balancing scheduling algorithms for wireless sensor networks". In: *CTRQ 2011, The Fourth International Conference on Communication Theory, Reliability, and Quality of Service.* 2011, pp. 54–59.

[C9] Janos Levendovszky, **Endre László**, Kalman Tornai, and Gergely Treplan. "Optimal pricing based resource management". In: *Proceedings of the International Conference on Operations Research* (2010), p. 169.

[LN1] Zoltán Nagy, Péter Szolgay, András Kiss, and **Endre László**. "GPU architektúrák". In: *Párhuzamos számítógép architektúrák, processzortömbök.* Pázmány Egyetem eKiadó, 2015. Chap. 3, pp. 34–59.

References

[1] Gene M. Amdahl. “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities”. In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference.* AFIPS ’67 (Spring). Atlantic City, New Jersey: ACM, 1967, pp. 483–485. DOI: 10.1145/1465482.1465560. URL: http://doi.acm.org/10.1145/1465482.1465560.

[2] John L. Gustafson. “Reevaluating Amdahl’s Law”. In: *Communications of the ACM* 31 (1988), pp. 532–533.

[3] Samuel Williams, Andrew Waterman, and David Patterson. “Roofline: An Insightful Visual Performance Model for Multicore Architectures”. In: *Commun. ACM* 52.4 (Apr. 2009), pp. 65–76. ISSN: 0001-0782. DOI: 10.1145/1498765.1498785. URL: http://doi.acm.org/10.1145/1498765.1498785.

[4] Krste Asanovic et al. “A View of the Parallel Computing Landscape”. In: *Commun. ACM* 52.10 (Oct. 2009), pp. 56–67. ISSN: 0001-0782. DOI: 10.1145/1562764.1562783. URL: http://doi.acm.org/10.1145/1562764.1562783.

[5] Yushan Wang, Marc Baboulin, Jack Dongarra, Joël Falcou, Yann Fraigneau, and Olivier Le Maître. “A Parallel Solver for Incompressible Fluid Flows”. In: *Procedia Computer Science* 18 (2013). 2013 International Conference on Computational Science, pp. 439–448. ISSN: 1877-0509. DOI: http://dx.doi.org/10.1016/j.procs.2013.05.207. URL: http://www.sciencedirect.com/science/article/pii/S1877050913003505.

[6] Tobias Brandvik and Graham Pullan. “An Accelerated 3D Navier–Stokes Solver for Flows in Turbomachines”. In: *Journal of Turbomachinery* 133.2 (2011), pp. 021025+. DOI: 10.1115/1.4001192. URL: http://dx.doi.org/10.1115/1.4001192.

[7] Thomas H. Pulliam. “Implicit solution methods in computational fluid dynamics”. In: *Applied Numerical Mathematics* 2.6 (1986), pp. 441–474. ISSN: 0168-9274. DOI: http://dx.doi.org/10.1016/0168-9274(86)90002-4. URL: http://www.sciencedirect.com/science/article/pii/0168927486900024.

[8] Thomas H Pulliam. “Solution methods in computational fluid dynamics”. In: *Notes for the von Kármán Institute For Fluid Dynamics Lecture Series* (1986).

[9] I.J.D. Craig and A.D. Sneyd. “An alternating-direction implicit scheme for parabolic equations with mixed derivatives”. In: *Computers and Mathematics with Applications* 16.4 (1988), pp. 341–350. ISSN: 0898-1221. DOI: http://dx.doi.org/10.1016/0898-1221(88)90150-2. URL: http://www.sciencedirect.com/science/article/pii/0898122188901502.

[10] Duy M. Dang, Christina Christara, and Kenneth R. Jackson. “Parallel Implementation on GPUs of ADI Finite Difference Methods for Parabolic PDEs with Applications in Finance”. In: *Social Science Research Network Working Paper Series* (Apr. 3, 2010). URL: http://ssrn.com/abstract=1580057.

[11] Craig C. Douglas, Sachit Malhotra, and Martin H. Schultz. *Parallel Multigrid with ADI-like Smoothers in Two Dimensions.* 1998.

[12] B. Düring, M. Fournié, and A. Rigal. “High-Order ADI Schemes for Convection-Diffusion Equations with Mixed Derivative Terms”. English. In: *Spectral and High Order Methods for Partial Differential Equations - ICOSAHOM 2012.* Ed. by Mejdi Azaiez, Henda El Fekih, and Jan S. Hesthaven. Vol. 95. Lecture Notes in Computational Science and Engineering. Springer International Publishing, 2014, pp. 217–226. ISBN: 978-3-319-01600-9. DOI: 10.1007/978-3-319-01601-6_17. URL: http://dx.doi.org/10.1007/978-3-319-01601-6_17.

[13] Samir Karaa and Jun Zhang. “High order ADI method for solving unsteady convection–diffusion problems”. In: *Journal of Computational Physics* 198.1 (2004), pp. 1–9. ISSN: 0021-9991. DOI: http://dx.doi.org/10.1016/j.jcp.2004.01.002. URL: http://www.sciencedirect.com/science/article/pii/S002199910400018X.

[14] D. W. Peaceman and Jr. Rachford H. H. “The Numerical Solution of Parabolic and Elliptic Differential Equations”. English. In: *Journal of the Society for Industrial and Applied Mathematics* 3.1 (1955), ISSN: 03684245. URL: http://www.jstor.org/stable/2098834.

[15] J. Douglas and H. H. Rachford. "On the numerical solution of heat conduction problems in two and three space variables". In: *Transaction of the American Mathematical Society* 82 (1956), pp. 421–489.

[16] Jr. Douglas Jim and JamesE. Gunn. "A general formulation of alternating direction methods". English. In: *Numerische Mathematik* 6.1 (1964), pp. 428–453. ISSN: 0029-599X. DOI: 10.1007/BF01386093. URL: http://dx.doi.org/10.1007/BF01386093.

[17] T. Roska and L.O. Chua. "The CNN universal machine: an analogic array computer". In: *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on* 40.3 (Mar. 1993), pp. 163–173. ISSN: 1057-7130. DOI: 10.1109/82.222815.

[18] Balázs Gergely Soós, Ádám Rák, József Veres, and György Cserey. "GPU Boosted CNN Simulator Library for Graphical Flow-based Programmability". In: *EURASIP J. Adv. Signal Process* 2009 (Jan. 2009), 8:1–8:11. ISSN: 1110-8657. DOI: 10.1155/2009/930619. URL: http://dx.doi.org/10.1155/2009/930619.

[19] Zsolt Vörösházi, András Kiss, Zoltán Nagy, and Péter Szolgay. "Implementation of embedded emulated-digital CNN-UM global analogic programming unit on FPGA and its application". In: *International Journal of Circuit Theory and Applications* 36.5-6 (2008), pp. 589–603. ISSN: 1097-007X. DOI: 10.1002/cta.507. URL: http://dx.doi.org/10.1002/cta.507.

[20] Christopher P Stone, Earl PN Duque, Yao Zhang, David Car, John D Owens, and Roger L Davis. "GPGPU parallel algorithms for structured-grid CFD codes". In: *Proceedings of the 20th AIAA Computational Fluid Dynamics Conference*. Vol. 3221. 2011.

[21] Joakim Beck and Serge Guillas. "Sequential design with Mutual Information for Computer Experiments (MICE): emulation of a tsunami model". In: *arXiv preprint arXiv:1410.0215* (2014).

[22] Dimitra Makrina Salmanidou, Aggeliki Georgiopoulou, Serge Guillas, and Frederic Dias. "Numerical Modelling of Mass Failure Processes and Tsunamigenesis on the Rockall Trough, NE Atlantic Ocean". In: *Proceedings of the Twenty-fifth (2015) International Ocean and Polar Engineering Conference* (2015).