Strategies for Improving Performance and Energy Efficiency on a Many-core

Elkin Garcia
Department of Electrical and Computer Engineering
University of Delaware
egarcia@udel.edu

Guang Gao
Department of Electrical and Computer Engineering
University of Delaware
ggao@capsl.udel.edu

ABSTRACT

New many-core architectures are characterized not only by the large amount of processing elements but also by the large number and heterogeneity of resources. This new environment has prompted the development of new techniques that seek finer granularity and a greater interplay in the sharing of resources.

The research proposed here will provide an analysis of these new scenarios, proposing new methodologies and solutions that leverage these new challenges in order to increase the performance and energy efficiency of modern many-core architectures.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Modeling techniques]; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel programming*; D.3.4 [**Processors**]: Optimization; G.1.0 [**Numerical Analysis**]: General—*Parallel algorithms*

General Terms

Algorithms, Performance, Theory

Keywords

Energy efficiency, Many-core architectures, Performance optimizations, Power-aware transformations

1. MOTIVATION AND SIGNIFICANCE

The new many-core era motivated by the recent efforts to build peta-scale and exa-scale machines has brought several challenges for exploiting the parallelism on new many-core architectures with hundreds, or even thousands, of independent processing elements. The scenario inside these chips is different to previous multi-core processors, some of the new characteristics are: (1) Increasing amount of shared resources, (2) heterogeneity of resources, (3) diversity in coordination and arbitration mechanisms for shared resources and (4) constraints in energy consumption.

This new environment requires new techniques that seek finer granularity and a greater interplay in the sharing of resources. These work re-evaluate several elements of computer systems and algorithm design under these new scenarios, it includes runtime systems, scheduling schemes and compiler transformations.

Moore's law is still valid, the number of transistor in a single chip doubles every 18 months approximately, but single processor architectures are not able to take advantage of the increasing amount of transistors. Today, Computer Architecture has become extremely parallel at all levels. The many-core era has arisen: A large number of simple processing elements are preferred over few very complex but powerful processors. The whole system is moving towards more heterogeneity with both approaches, where the share resources are essential for a proper synergy.

2. PROBLEM FORMULATION AND METHODOLOGY

This new era brings two main challenges in the algorithms implemented on these modern many-core architectures: First, shared resources have become the norm, ranging from the memory hierarchy and the interconnections between processing elements and memory to arithmetic blocks such as double floating point units; different mechanism at software and hardware levels are used for the arbitration of these shared resources and need to be consider on the scheduling and orchestration of tasks. Second, in order to take advantage of the increasing amount of parallelism available, the number of tasks has increased and tasks have become finer, imposing new challenges for a light and balanced scheduling subject to resource and energy constrains.

The research proposed here will provide an analysis of these new scenarios, proposing new methodologies and solutions that leverage these new challenges in order to increase the performance and energy efficiency of modern many-core architectures. During the pursue of these objectives, this research intends to answer the following question:

- Which is the impact of low-level compiler transformations such as tiling and percolation to effectively produce high performance code for many-core architectures?
- What are the trade-offs of static and dynamic scheduling techniques to efficiently schedule fine grain tasks with hundreds of threads sharing multiple resources under different conditions in a single chip?
- How to efficiently model energy consumption on many-cores managing trade offs between scalability and accuracy?

- Which are feasible methodologies for designing power-aware tiling transformations on many-core architectures?

The research methodology adopted includes theoretical analysis and experimental evaluation. Selected analysis and results presented in these work have been extended on our previous publications [9, 6, 7, 8]. For the evaluation, we have used a novel many-core architecture: The IBM Cyclops-64 processor (C64), a double precision architecture with 160 independent hardware thread units on a chip and a software managed memory hierarchy. C64 features have been described extensively in previous publications [4].

3. PROPOSED APPROACH

3.1 Performance

Methodologies for improving performance in parallel systems have been focused on cache-based parallel systems. This methodologies exploit locality through padding and cache tiling techniques with tile size selection [3, 12]. Nevertheless, cache replacement policies are controlled by hardware, making fine control of these parameters difficult. A feasible alternative is the new set of many-core-on-a-chip systems with a software managed memory hierarchy such as Cyclops-64 (C64). These new systems provide more opportunities to improve performance and flexibility.

We present a general framework that provides a mapping of applications to software managed memory hierarchies, our strategy involves an optimal register tiling and sequence of traversing tiles. The size of tiles and the sequence of traversing tiles are designed to maximize the reuse of data in registers and minimize the number of memory accesses to slower levels by solving a non-linear optimization problem [9].

At task level, we put special attention to the case of fine-grain parallelism. In the past, the main focus of scheduling techniques was to achieve high load balancing with low overhead in order to increase total performance. As a result, Static Scheduling (SS) is preferred over Dynamic Scheduling (DS) for embarrassingly parallel and highly regular applications executed in homogeneous architectures.

We have studied the task scheduling problem for this kind of applications under the scenario imposed by many-core architectures to investigate whether or not there exists scenarios where DS is better than SS. We found that for highly regular and embarrassingly parallel applications, DS can overcome SS in some situations commonly found in many-core architectures. We present experimental evidence that support this claim. In particular, we show how the performance of SS is degraded by the new environment on many-core chips [8, 5, 13]. We analyze three reasons that contribute to the poor performance of SS and the feasibility of a light DS implementation on many-core architectures under the situations described: (1) Under limited amounts of tasks, a uniform mapping of work to processors without considering the granularity of tasks is not necessarily scalable. (2) The presence of shared resources (i.e. shared memory and crossbar interconnections) produces unexpected variations in the task execution time of similar sizes. SS is not able to manage these variations properly. (3) Hardware features, such as in-memory atomic operations, greatly contribute to decrease the overhead of DS, making it competitive with respected to SS.

In addition, we have study the role of task percolation and scheduling in the performance of applications. In order to increase the returns of these techniques we have made two improvements. First, we fused dynamic scheduling and percolation operations into a *dynamic percolation* approach. Second, we propose to add additional percolation operations. Our new improvements contribute to raise the performance of certain applications in many-cores. We used the Matrix Multiply benchmark under the C64 architecture. For this example, we increase the performance by 48% [7, 5].

3.2 Energy Efficiency

On modern parallel architectures, power consumption and energy efficiency are a feasible constraint. In this area, we propose a scalable energy consumption model for many-core architectures with software-managed memory hierarchy. In addition we develop a power-aware tiling techniques for energy friendly applications [6].

The proposed energy consumption model is function of two parameters: The total execution time and the number and type of instructions executed in the application. In more detail, for a program Λ, the total energy consumption E_T can be expressed by:

$$E_T(\Lambda) = e_0 \cdot t + \sum_{i=1}^{M} e_i \cdot N(C_i) \qquad (1)$$

Where e_0 is the static power dissipated, and e_i for $i = 1, \ldots, M$ is the energy consumed by each type of instruction C_i (e.g. integer addition, double multiplication, load double from on-chip memory, etc). The function $N(\cdot)$ counts the number of instructions in the program Λ that belong to that type of instruction. The simplicity of this model make it scalable with the number of hardware threads. In addition, the model considers variations in latency of the operations produced by diverse mechanism of arbitration in shared resources and starvation of them.

Finally, the design of the power-aware tiling follows an approach supported by the solution of a non-linear optimization problem that minimizes the energy consumption of the most power hungry operations. This optimization problem is targeting to minimize the dynamic energy, this portion is close related to the amount of work performed, and it does not depend on how the work is executed (serial or parallel). While latency can be hidden by proper overlapping of tasks in order to decrease execution time and increase performance, performance optimizations are just decreasing the static energy consumption but can positively or negatively impact the dynamic energy consumption.

We have study different approaches to solve this optimization problem under different applications such as FDTD, LU decomposition and matrix multiplication. We have validated our model on a real C64 chip, showing high accuracy. Also the power-aware tiling has shown success reducing the total energy consumption and increasing the power efficiency.

4. EXPERIMENTAL RESULTS

4.1 Performance

First, we used a microbenchmark to evaluate SS vs DS. The tasks in this microbechmark process 256B of data from on-chip memory as follows: The thread copies a vector from on-chip memory to local memory. Then, it computes the checksum of the bytes and it stores back the vector in another location of the memory. Figure 1 (left) shows the trade offs between DS and SS with different number of Thread Units (TUs). As expected, when few TUs are used, SS is faster than DS, with the worst case being -27% for 32 TUs. After 24 TUs, DS has better performance when few task are used. Finally, after 48 TUs all the datasets favor DS. The maximum relative speed up is 137%. This speed up occours with the smallest problem size and the maximum number of TUs.

Using Matrix Multiplication and on-chip memory, our baseline used optimized register tiling and SS, its scalability with the matrix size can be detailed in Figure 1 (center). We further increased the performance to 58.95 GFLOPS by using the Percolation inside the tasks using the same SS but the scalability remains the same. With the implementation of DS, the maximum performance and scalability with respect to the number of TUs increased significantly.

A hierarchical tiling for matrix multiply was also studied. We used large matrices in off-chip memory using Percolation for the overlapping of computation and data movement. We compared two versions of the matrix multiplication: The first one is a static scheduling approach that assigns computation and data movement tasks at the beginning of execution using low latency hardware barriers for synchronization between tasks. The second one is dynamically scheduled; the tasks are ready after their dependencies are satisfied [5]. We have used a Dynamic Percolation that takes advantage of the in-memory atomic operations available in C64. The results in figure 1 (right) show the high scalability and performance of the Dynamic Scheduling approach. On the contrary, the Static version is not able to surpass half the theoretical peak performance of C64. In addition, the SS scalability decreases after 120 TUs.

4.2 Energy Efficiency

Using measurements of instantaneous power on a real chip and a regression model, the coefficients of equation 1 were calculated. The results for a subset of the Instruction Set of C64 are shown on Figure 2 (left). From our analysis, loads and stores on DRAM (ldddram, stddram) are the most energy hungry, followed by loads and stores on SRAM (lddsram, stdsram). There is a difference of two orders of magnitude in the energy consumption of DRAM operation and SRAM operations. A deeper analysis using several microbenchmarks show that the energy consumption is highly linear with the number of operations of each type. It details that after memory operations, floating point operations (fmaddd, fmuld and fmad) and complex integer operations (mull) consumes similar energy. Integer, logical and register movement operations (add, and, mov, li) consumes less energy than the other classes.

We also evaluated our energy-aware tiling using two applications: Matrix Multiply (MM) and Finite Difference Time Domain (FDTD). First, we will compare real measurements of energy consumption with the estimations made using our model in order to determine the accuracy of it. Next, we will compare different tilings with the one designed for been energy-aware in order to study the advantages of the technique proposed.

A matrix of size $m \times m$ that fits on SRAM was used for the MM benchmark. Our approach *(OptT)* was compared with a dot product register tiling *(DPT)*. Both tilings were implemented using assembly code in order to use the whole register file. The FDTD implementation used a problem of size $m \times q$, the tile size is the maximum possible that fits on Scratchpad Memory, we compare our diamond tiling *(DmT)* with 3 well-known techniques: A rectangular tiling (naive) *(NT)*, the overlapped tiling *(OT)* that uses redundant computations in order to tile time and space dimensions and split tiling *(ST)* that uses multiple shapes for fully partitioning the iteration space.

Figure 2 (center) compares the measured energy with the prediction of our model using the Matrix Multiplication. Clearly, the predictions have high accuracy to the measurements for the dynamic energy and static energy. The average error of our model is 26.6% and 0.82% for the dynamic energy and total energy respectively. In addition, the tiling proposed decreases the dynamic and total energy consumption in 56.52% and 61.21% on average. Further calculations show that the power efficiency [MFLOPS/W] increases between 2.62 and 4.13 times for this benchmark.

For the FDTD benchmark, figure 2 (right) shows the effectiveness of our energy-aware tiling (diamond tiling) for decreasing the total and dynamic energy. The total average energy reduction was 81.26%, 57.27% and 15.69% compared with split tiling, overlapped tiling and naive tiling respectively. The accuracy of our model is corroborated too; the average error is 7.3% for the total energy with respect to real measurements.

5. RELATED WORK

Studies for increasing performance on many-core architectures with software-managed memory hierarchy and hundreds of independent threads on a single chip have been of recent interest [9, 2, 11], several approaches used for cache based systems have been able to increase performance but they are still far to take full advantage of the hardware and reach near peak performance even for highly regular applications [2, 11]. Some of them have shown empirical evidence about increasing the power efficiency [9, 5].

Under traditional architectures, energy consumption has been extensively studied [14]. Most of the research in the area has focused on systems with caches [10]. Models with high accuracy but also highly complex have been proposed for serial processors. They uses precise information about the hardware and they are based on elaborated instruction scheduling [14]. Extensions of these models to many-core architectures is highly difficult and not scalable with the number of hardware threads. Energy efficiency on multiprocessors has been focused on the hardware design, including hardware features like power saving off-chip memory or dynamic voltage selection [1].

6. FUTURE WORK

Several studies on regular applications have been done. A more detailed studies on irregular applications and the advantages of DS and Dynamic Percolation will allow more general conclusions. At this point, tilings for high performance and energy efficiency have been designed independently. Future work is pointing to formulate and solve an optimization problem for finding a tiling that includes trade offs between energy efficient and high performance.

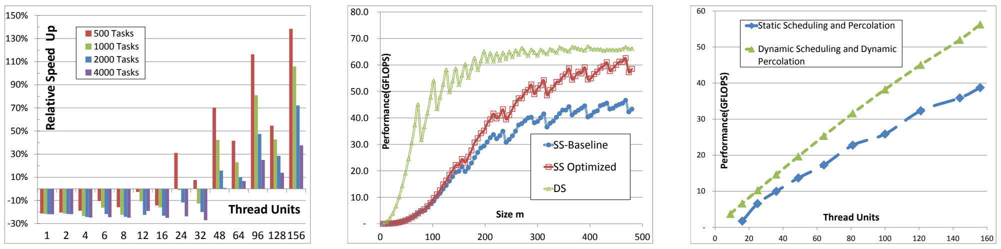

Figure 1: Results for Performance. *LEFT*: Relative Speed Up of DS vs. SS of Memory Copy. *CENTER*: Scalability for a MM in SRAM with 144 Threads. *RIGHT*: Scalability for a MM of size 6480 × 6480

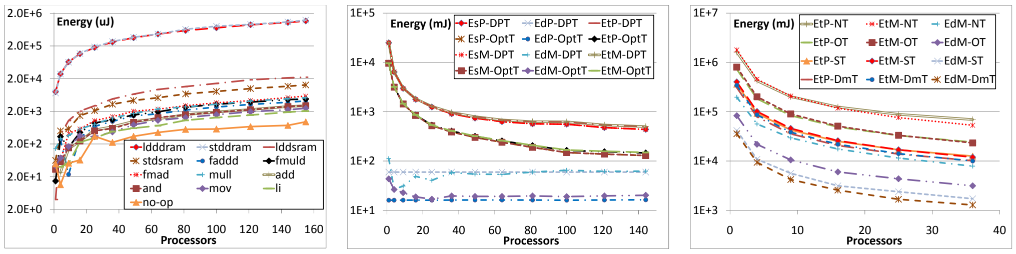

Figure 2: Results for Energy Efficiency. *LEFT*: Overall comparison of selected ISA. *CENTER*: Energy consumption (Static Es, Dynamic Ed and Total Et) vs Predicted model P and Measured M using different tilings for MM with $m = 300$. *RIGHT*: Energy consumption (Static Es, Dynamic Ed and Total Et) vs Predicted model P and Measured M using different tilings for FDTD with $m = 100k$ and $q = 500$

7. ACKNOWLEDGEMENTS

This work has been made possible by the generous support of the NSF through research grants CCF-0833122, CCF-0925863, CCF-0937907, CNS-0720531, and OCI-0904534.

It was also partly supported by European FP7 project TERAFLUX, id. 249013.

8. REFERENCES

[1] Andrei, A., Eles, P., Peng, Z., Schmitz, M., Hashimi, B.: Energy optimization of multiprocessor systems on chip by voltage selection. Trans. VLSI 15(3), 262 –275 (mar 2007)

[2] Chen, L., Hu, Z., Lin, J., Gao, G.R.: Optimizing the Fast Fourier Transform on a Multi-core Architecture. In: IPDPS '07. pp. 1–8 (Mar 2007)

[3] Coleman, S., McKinley, K.S.: Tile size selection using cache organization and data layout. In: PLDI'95. pp. 279–290. ACM, New York, NY, USA (1995)

[4] del Cuvillo, J., Zhu, W., Hu, Z., Gao, G.R.: FAST: A Functionally Accurate Simulation Toolset for the Cyclops-64 Cellular Architecture. In: MoBS'05. pp. 11–20 (2005)

[5] Garcia, E., Khan, R., Livingston, K., Venetis, I.E., Gao, G.R.: Dynamic percolation - mapping dense matrix multiplication on a many-core architecture. CAPSL TM098 (June, 2010)

[6] Garcia, E., Orozco, D., Gao, G.: Energy efficient tiling on a Many-Core Architecture. In: MULTIPROG-2011. pp. 53–66. Heraklion, Greece (January 2011)

[7] Garcia, E., Orozco, D., Khan, R., Venetis, I., Livingston, K., Gao, G.R.: Dynamic Percolation: A case of study on the shortcomings of traditional optimization in Many-core Architectures. In: CF 2012. ACM, Cagliari, Italy (May 2012)

[8] Garcia, E., Orozco, D., Pavel, R., Gao, G.R.: A discussion in favor of Dynamic Scheduling for regular applications in Many-core Architectures. In: MTAAP 2012. IEEE, Shanghai, China (May 2012)

[9] Garcia, E., Venetis, I.E., Khan, R., Gao, G.: Optimized Dense Matrix Multiplication on a Many-Core Architecture. In: Euro-Par 2010. LNCS, vol. 6272, pp. 316–327. Springer-Verlag, Ischia, Italy (August 2010)

[10] Hanson, H., Hrishikesh, M., Agarwal, V., Keckler, S., Burger, D.: Static energy reduction techniques for microprocessor caches. Tran. VLSI 11(3), 303 – 313 (jun 2003)

[11] Hu, Z., del Cuvillo, J., Zhu, W., Gao, G.R.: Optimization of Dense Matrix Multiplication on IBM Cyclops-64: Challenges and Experiences. In: Euro-Par 2006. pp. 134–144. Dresden, Germany (Aug 2006)

[12] M. D. Lam, E.E.R., Wolf, M.E.: The cache performance and optimizations of blocked algorithms. In: ASPLOS-IV. pp. 63–74. ACM, New York, NY, USA (1991)

[13] Orozco, D., Garcia, E., Pavel, R., Gao, G.: TIDeFlow: The Time Iterated Dependency Flow Execution Model. In: Proceedings of Workshop on Data-Flow Execution Models for Extreme Scale Computing (DFM 2011); 20th International Conference on Parallel Architectures and Compilation Techniques (PACT 2011). pp. 1–9. IEEE Computer Society, Galveston Island, TX, USA (October 2011)

[14] Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced cpu energy. pp. 374 –382 (oct 1995)