Landing Stencil Code on Godson-T*

Huimin Cui[1,2], Lei Wang[1,2], Dongrui Fan[1], and Xiaobing Feng[1]

[1] Key Laboratory of Computer System and Architecture, Institute of Computing Technology, CAS, 100190 Beijing, China

[2] Graduate University of Chinese Academy of Sciences, Beijing 100039, P. R. China

E-mail: {cuihm, wlei, fandr, fxb}@ict.ac.cn

Abstract The advent of multi-core/many-core chip technology offers both an extraordinary opportunity and a profound challenge. In particular, computer architects and system software designers are faced with a unique opportunity to introducing new architecture features as well as adequate compiler technology – together they may have profound impact.

This paper presents a case study (using the 1D Stencil computation) of compiler-amendable performance optimization techniques on a many-core architecture Godson-T. Godson-T architecture has several unique features that are chosen for this study: (1) chip-level global addressable memory – in particular the scratchpad memories (SPM) local to the processing cores; (2) fine-grain memory based synchronization (e.g. full-empty bit for fine-grain synchronization).

Leveraging state-of-the-art performance optimization methods for 1-D stencil parallelization (e.g. timed tiling and variants), we developed and implement a number many-core based optimization for Godson-T. Our experimental study show good performance improvements in both execution time speedups and scalability, validated the value of globally accessed SPM and fine-grain synchronization mechanism (full-empty bits) under the Godson-T, and provide some useful guidelines for future compiler technology of many-core chip architectures.

Keywords: many-core, stencil, Jacobi, compiler, SPM, fine-grain synchronization

1 Introduction

High-performance processor design is rapidly moving towards many-core architectures that integrate 10s (or beyond) of cores on a single chip [1,2]. Intel recently announced Larrabee, a many-core x86 architecture [3]. IBM Cyclops-64 will support 160 hardware thread units in one chip [4,5]. Many-core architecture offers opportunities and challenges to architects and system software designers. Their cooperation of new architecture features design would have profound impact.

Stencil computations represent a practically important class of computations that arise in many

* Supported by the National Grand Fundamental Research 973 Program of China (No. 2005CB321602), National Natural Science Foundation of China (No. 60736012), National High Technology Research and Development Program of China (No. 2007AA01Z110), and the National High-Tech Research and Development Plan of China (No.2009AA01Z103).

scientific/engineering codes. Computational domains that involve stencils include those that use explicit time-integration methods for numerical solution of partial differential equations, and multimedia/image-processing applications that perform smoothing and other neighbor pixel based computations [6]. There has been some prior work that has addressed [6,7,8,9,10,11,12,13,14,15].

In this paper, we develop and implement a number many-core based optimization for Godson-T, leveraging state-of-the-art performance optimization methods for 1-D stencil parallelization (e.g. timed tiling and variants). Our experimental study show good performance improvements in both execution time speedups and scalability, validated the value of globally accessed SPM and fine-grain synchronization mechanism (full-empty bits) under the Godson-T, and provide some useful guidelines for future compiler technology of many-core chip architectures.

The paper is organized as follows. Section 2 overviews Godson-T. Section 3 introduces our motivation example and states our problem. Section 4 presents our methods. Section 5 shows our experimental results, and section 6 and 7 presents the related work and conclusions respectively.

2 Godson-T Architecture

Godson-T [16,17,18,19] is a processor prototype of many-core chip. Figure 2.1 gives the overview of the Godson-T processor architecture. It has 64 homogeneous, in-order and dual-issue processing cores. The target frequency of each core is 1GHz.

Each processing core has a 32KB local memory. It is private L1 cache by default, and can be configured as an explicitly-controlled Scratchpad Memory (SPM), or a hybrid of cache and SPM. SPM can be globally accessed, and the bandwidth is 256GB/s. It provides low-latency access, that is, 1 cycle for local SPM access, and 2 cycles/hop for remote SPM access. In addition, Godson-T has shared L2 cache.

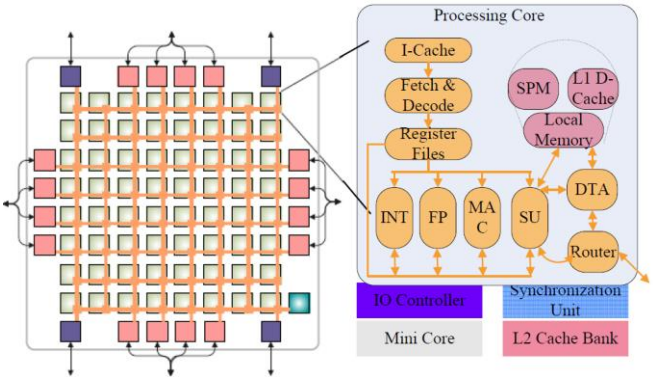


Fig. 2.1 Overview of Godson-T

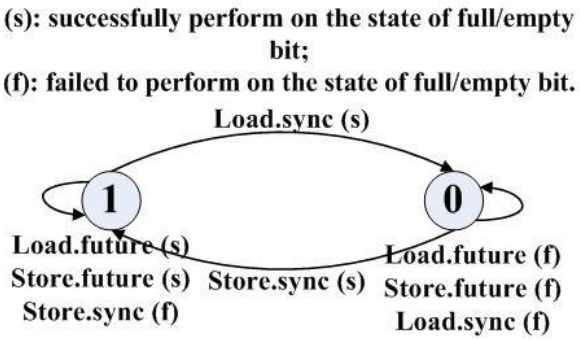


Fig.2.2 State machine for full-empty bit of Godson-T

When configured as SPM, 8 bits of tag a cacheline become word-level full-empty bit for fine-grain synchronization. The full-empty bit tagged on memory cell indicates the presence of data on the memory location, e.g. “1” for “full”, and “0” for “empty” [20]. There are two types of fine-grain synchronization instructions: sync type and future type. The former is used for producer-consumer style synchronization, whereas

the latter is used for future data object protection. Figure 2.2 illustrates the state machine of full-empty bit.

3 Motivation Example

3.1 1-D Jacobi

At each time *t*, 0 < *t* < T, a stencil computation updates a grid point based on the values of the point and some neighboring points to produce the value of the point at time *t+1* [7]. The simplest stencil computation is 1-D Jacobi. It updates a point using the average of it and its neighbors, and the code is shown in Figure 3.1 [21]. For simplicity of explanation, the code is rewritten to Figure 3.2 [6].

```
for t = 0 to T-1
 for i = 1 to N-2
  B[i] = (A[i-1]+A[i]+A[i+1])/3;
  for i = 1 to N-2
   A[i] = B[i];
```

Fig.3.1 1-D Jacobi code

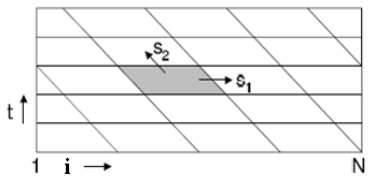
```
 for t = 1 to T-1
  for i = 1 to N-2
  A[t, i] =  (A[t-1,i-1]+
  A[t-1,i]+A[t-1,i+1])/3;
```

Fig.3.2 Single statement form of 1-D Jacobi

3.2 Existing Methods

1-D Jacobi's performance would be limited by memory bandwidth if naïve implemented [22]. To reduce memory traffic, leverage tiling in both the spatial and temporal dimensions uses loop skewing in order to increase data reuse. Studies by McCalpin [11] and others [12, 13] have shown time skewing can benefit [8]. Time skewing involves loop skewing and tiling [13], and tiles of shape are shown in Figure 3.3(a) which was called standard tiling [6]. The horizontal axis represents the spatial dimension and the vertical is time dimension. Figure 3.3(b) shows the execution order with pseudo-code. We can see that the tiles are started in a pipeline manner.

On multi-core/many-core architecture, standard tiling must tradeoff between achieving good data reuse and load balance of parallel execution [6]. So overlapped and split tiling are presented in [6].

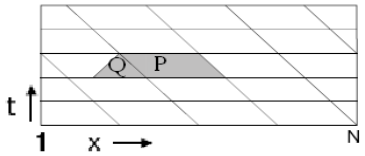


(a) tile shape

```
for (lt =0; lt <i_tiles + t_tiles ;
lt++) {
  forall tiles indexed as (it, tt)
     if (it + tt == lt)
       compute tile(it, tt)
}
```

(b) pseudo-code to show the execution order

Fig.3.3 standard tiling of 1-D Jacobi, time skewing



(a) tile shape

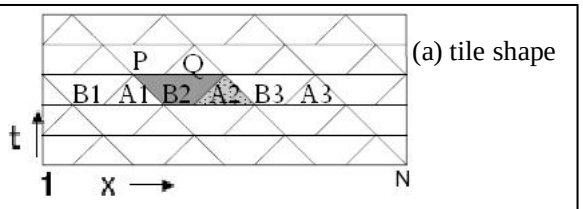
```
for (tt = 0; tt < t_tiles ; tt++) {
 forall (it=0; it < i_tiles, it++) {
    generate local new_tile using
tile(tt, it) and tile(tt, it-1);
    compute new_tile;
  }
}
```

(b) pseudo-code to show the execution order

Fig.3.4 Overlapped tiling of 1-D Jacobi [6].

Overlapped tiling is shown in Figure 3.4. An additional triangular region (Q) is added to the left of the tile P. It eliminates the dependence between tiles along the horizontal direction, and all the tiles belonging to the same time slice can start concurrently. The time slices are executed in sequential order.

But overlapped tiling has redundant computation, Q is calculated twice. To eliminate the redundancy, [6] presented split tiling, as shown in Figure 3.5. It splits each standard tile into two sub-tiles (Ai and Bi). At first step all non-dependent sub-tiles Ai are executed concurrently, and then the dependent sub-tiles Bi [6]. Similar with overlapped tiling, the order of time slices is kept sequential.



(a) tile shape

```
for (tt = 0; tt < t_tiles ; tt++) {
  forall (it = 0; it < i_tiles, it++)
    compute triangle_area A(it);
  barrier
  forall (it = 0; it<i_tiles, it++)
    compute trapezia_area B(it);
  barrier
}
```

(b) pseudo-code to show the execution order

Fig.3.5 Split tiling of 1-D Jacobi [6]

3.3 Some Open Questions and Problem Statement

There are three challenges (open questions) that need to be addressed for stencil running on Godson-T. We take Figure 3.5 to present these open questions. For clarification, we explicitly write barrier statements implied by forall clause.

How to keep good scalability with large number of cores? Muthu presented bad scalability of existing methods in [14]. This is critical for many-core architecture.

How to reduce the cost of barriers, and furthermore eliminate them? As we know, barrier is expensive [22]. In existing methods, all the up-to-date values need to be written to the shared L2 cache at barrier points. Can we reduce the cost?

Furthermore, In Figure 3.5, suppose Ai and Bi is executed by Pi sequentially. P2 can execute Q just after B2 is finished, and is unnecessary to wait for others. Therefore barrier is unnecessary. Can we find some method to totally eliminate them?

Would traditional optimizations be applicable for many-core, and how to determine the optimization parameters? Traditional optimizations are widely used in current compilers. But the effect and scalability is still a problem on many-core.

We state our problem as follows. Given a Godson-T like many-core architecture (with two features: globally accessed SPM and fine-grain memory based synchronization), how to land a stencil program such that the open issues (as outlined above) will be effectively addressed.

4 Optimizations on Godson-T

Our optimizations utilize two hardware features: globally accessed SPM and fine-grain synchronization. Similar but different mechanisms are supported on Cyclops64 [4]. We choose split tiling as our startup, since it has no redundancy.

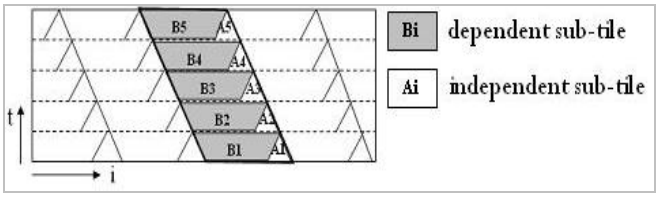
4.1 Overview of Our Method

To expose fine-grain synchronization chances, we must determine the tile schedule policy. We use static binding as our policy. As shown by Figure 4.1(a), each core executes a highlighted parallelogram. We use tile to represent it, and sub-tile for triangle (Ai) and trapezium (Bi) areas. In Figure 4.1(a), the highlighted tile contains 10

sub-tiles (A1~A5, B1~B5). The execution order of sub-tiles is shown in (b).

We can see that each core would use *xtilesize*ttilesize* spaces for calculating a tile (*xtilesize* and *ttilesize* represent space and time tile size respectively), which should not exceed the size of SPM. As the problem size grows, we use DTA [18] to overlap data transferring and computing, but it's beyond the scope of this paper.

We manually applied three major optimizations starting from naïve split tiling: (1) traditional optimizations for single thread; (2) using SPM to reduce barrier cost; (3) using fine-grain synchronization to eliminate barriers.



(a) tile shape with our method

```
forall (it = 0; it < i_tiles; it++)
  for (tt = 0; tt < t_tiles; tt++) {
    if (A(tt-1, it) has been passed out))
      compute A(tt, it);
    if (dependencies of B (tt, it) satisfied)
  compute B(tt, it)
}
```

(b) execution order of tiles and sub-tiles

Fig.4.1 Our method: split tiling without barriers

4.2 Traditional Optimizations

Using multiplication replacing division. Division has longer latency than multiplication, so we replace it by multiplication.

Loop unrolling + register allocation + instruction schedule + bottom loading. The unroll factor is decided by the number of registers. Register allocation and instruction scheduling are done after loop unrolling. Bottom loading is an effective technique for overlapping loop control and loading of operands for the next iteration of the loop [23].

Computation transformation and optimization (loop unrolling + common sub-expression elimination (CSE) + Using MADD + register allocation + software pipelining + instruction schedule). It seems we cannot use multiply-and-add (MADD), but it would turn around after some transformations. The original statement can be rewritten into *A[t,i]=A[t-1,i-1]*1/3+(A[t-,i]+A[t-1,i+1])*1/3* or *A[t,i]=(A[t-1,i-1]+A[t-1,i])*1/3+A[t-1,i+1]*1/3*.

Thus MADD instructions can be used. Furthermore, after loop unrolling, the calculation of A[t, i] and A[t, i+1] would have a common sub-expression *(A[t-1,i]+A[t-1,i+1])*1/3*, and can be eliminated by CSE.

4.3 Using SPM

As mentioned in section 2, Godson-T can be configured to use SPM instead of cache. In this configuration, SPMs are mapped to a consecutive global address space. Therefore, all SPMs can be easily accessed by either local or remote cores.

It is not difficult for the programmers to use SPM. The program needs minor modification, just declaring a pointer or array on SPM, and it can be used as ordinary memory. Programmers can move original data from off-chip memory to SPM, calculate, and then store the data to off-chip memory. Godson-T runtime system provides a clear interface for programmers.

Using SPM won't eliminate the barriers, but it reduces the cost of barriers. At the barrier point, the up-to-date data needn't to be written to the shared cache any more.

4.4 Using Fine-Grain Synchronization

Godson-T supports full-empty bit based on SPM. So this optimization must be applied after the optimization of using SPM is done. Fine-grain synchronization has been researched on many architectures: HEP [24], Tera [25], MDP [26], Alewife [27], M-Machine [28], Cray MTA-2 [29], and the start-of-the-art many-core Cyclops-64 [4].

Data Distribution. We take Figure 4.2 for presenting our data distribution. Each core allocates a buffer on its SPM with size of *ttilesize***xtilesize* to hold the points of a time slice. In Figure 4.2, Bi represents the black points and Ai the white points, i is the core id. Their storage has no isolation, which means Ai's *n*-th line exactly follows Bi's. At the beginning of each time slice, all the data are located at *buffer[ttilesize-1]*. The generated data are stored into *buffer[0]~buffer[ttilesize-1]* in sequential order with t increasing from 0 to *ttilesize*-1. The buffer would be used repeatedly for each time slice, for good data reuse.

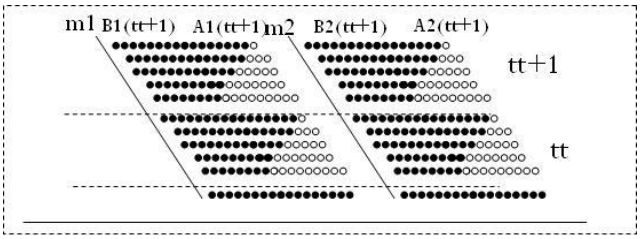


Fig.4.2 data distribution of 1-D Jacobi on Godson-T

Data Communication. Figure 4.3 shows the communication pattern. For each line, the two points of the right-side would be used by another core. Some would be used only once, while some twice.

It seems that we can use one-writer-multiple-reader synchronization as our implementation. But actually it doesn't work. Because the buffer is used repeatedly for each time slice, the next time slice must determine when a location can be rewritten. The new data cannot be written until the original value has been taken away. So this problem should be modeled as single-producer-multi-consumer.

Godson-T's full-empty bit only implements a single-producer-single-consumer model. Weirong presented synchronization state buffer in [4], using counter for multi-consumers. Our architecture designers are considering about this problem.

Without hardware support of multi-consumers, we can use a naïve method for stencil computation. At first step, using synchronized load operations to read the data from remote SPM into a temporary variable. And second step, using ordinary load instructions at each consuming point. Only 2**ttilesize* points need to be synchronized loaded and stored, which is a tiny portion of the computation set.

Things would turn more complex if the consumers belong to different tiles executed by different cores and the execution order is non-deterministic. That's beyond the scope of this paper.

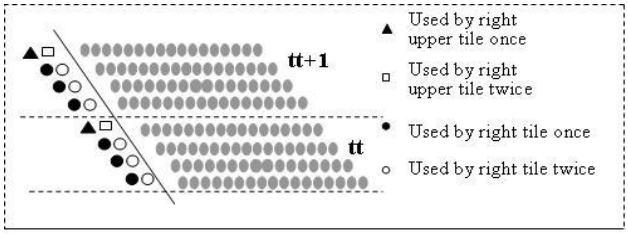


Fig.4.3 Communication pattern of 1-D Jacobi

5 Experimental Results

5.1 Experimental Framework

Our experiments are implemented on

Godson-T, which was introduced in Section 2. The program is compiled using the GCC compiler with –O3 option, and the kernel loop is written with assembly language. For the sake of comparison, we also implemented overlapped tiling.

Since the floating point registers of Godson-T is 32 bits, all the experiments are single floating point computations. We take FLOPS (floating point operations per second) as our performance evaluation. For 1-D Jacobi, in each iteration of *t*, the total computation is 3NT. 3NT/exec-time is the performance we can get.

5.2 Summary of Main Results

Our main experimental results can be summarized as follows.

Observation 1 (see also Section 5.3.1 for details): Our program optimization methods can be applied effectively to the stencil computation in the 1-D Jacobi code tested. Performance improvements on our experimental platform are observed in both execution time speedups and scalability.

Observation 2 (see also Section 5.3.2 for details): Our experimental results show the important value of optimization of using new hardware features. In particular, the globally accessed SPM and the fine-grain synchronization mechanism (full-empty bits) under the Godson-T many-core chip technology is studied and evaluated. The former reduces barrier cost, and the latter eliminates barriers totally.

Observation 3 (see also Section 5.3.3 for details): Our experimental results provide some useful guidelines for future compiler technology of many-core chip architectures. Our findings are three fold.

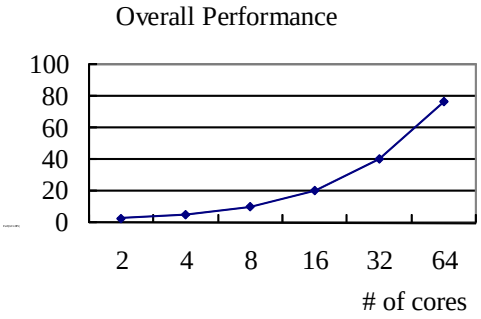
First, it appears that the major optimization methods reported in our work (such as efficiently usage of globally accessed SPM and fine-grain synchronization exploration) should be considered to be incorporated in future many-core compiler/tools to explore the performance advantages.

Second, some traditional compiler optimizations still play important roles on many-core platform – such as loop unrolling, register allocation, instruction scheduling, and software pipelining – their judiciary application are important to the overall performance.

Third, parameters for compiler optimizations can be statically determined, including the time tile size and space tile size. Space tile size can be chosen according to the size of SPM while time tile size can be chosen from a set of small integers.

5.3 Detailed Results and Analysis

5.3.1 Overall Performance



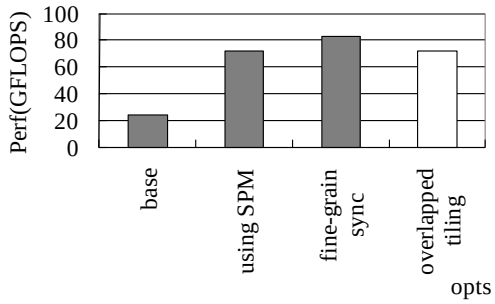
Program size: N = 128K, T = 256.
TTILESIZE = 4
Number of cores: varies from 2 to 64.

Fig.5.1 Performance and scalability of overall performance

Figure 5.1 shows the overall performance we can get when we vary the number of cores from 2

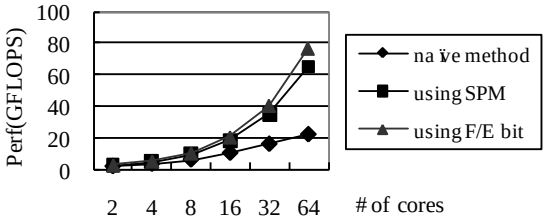
to 64. It shows that the performance can be almost doubled as the number of cores is doubled, and we get almost linear performance as the number of cores increases (the horizontal axis uses logarithmic scale, so it's a curve).

5.3.1 SPM and Fine-grain Synchronization's Effects



Program size: N = 128K, T = 1024.
Number of cores: 64; TTILESIZE = 4

Fig.5.2 Performance of our major opts. (using SPM and fine-grain synchronization)



Program size: N = 128K, T = 256.
Number of cores: varies from 2 to 64. *ttilesize* = 4

Fig.5.3 Scalability of our major optimizations (using SPM and fine-grain synchronization)

First, we fix the number of cores to 64, Figure 5.2 shows the performance. Before using SPM and fine-grain synchronization, we can get the performance of 24.13GFLOPS (with traditional optimization done, and its contribution would be evaluated in section 5.3.3). Using SPM reduces the barrier cost, and the performance improves to 72.12GFLOPS. Finally, barrier is eliminated by fine-grain synchronization, and we get the performance of 82.98GFLOPS. The best performance using overlapped tiling is also shown in Figure 5.2 for comparison.

Second, we evaluate the scalability of our major optimizations. Figure 5.3 shows the effects of our optimizations as number of cores varies from 2 to 64. Each line represents the performance scalability when adding a new optimization. We can see that each newly-added optimization provides good scalability.

5.3.1 Guidelines for Compiler

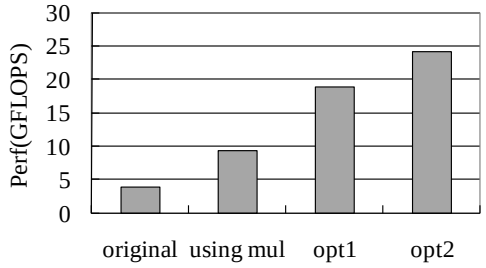
Section 5.3.2 shows the value of our major optimizations, including efficiently usage of globally accessed SPM and fine-grain synchronization exploration. These optimizations should be considered in future many-core compilers for performance.

Figure 5.4 shows the performance improvement with traditional optimizations applied. The performance is improved to 24.13GFLOPS from 3.77GFLOPS. And Figure 5.5 shows the traditional optimizations' scalability.

Time tile size is an important parameter during our optimization. In previous methods, larger time tile size would bring better data reuse and benefit. But in our method, SPM is used repeatedly as a rotating buffer, and barrier is eliminated. So it's unnecessary to choose a large time tile size any more.

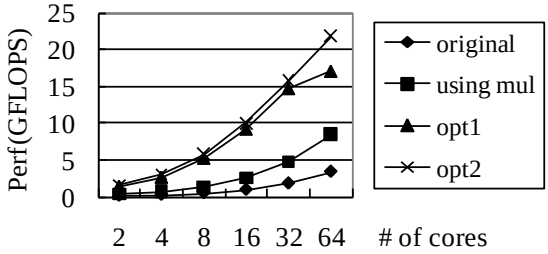
Figure 5.6 shows the performance with time tile size as 2, 4 and 8. We can see that time tile size can be fixed as a small value set, e.g. 2 and 4. Increasing time tile size doesn't benefit. Meanwhile, it causes the space tile size decreasing, since *ttilesize*xtilesize* cannot exceed the size of

SPM. So the performance declines as time tile size increases.



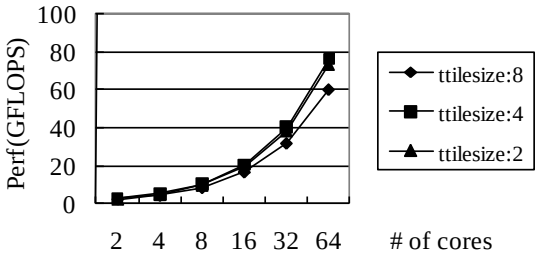
Opt 1: loop unrolling + register allocation + instruction schedule + bottom loading
Opt 2: computation transformation + loop unrolling + madd instruction + register allocation + instruction scheduling + software pipelining + CSE

Fig.5.4 Traditional optimizations' contributions



Program size: N=128K, T= 256.
Number of cores: 64. TTILESIZE = 4.
opt1/opt2 same with Fig 5.4

Fig.5.5 Traditional optimizations' scalability



Performance has no difference when time tile size is 2 and 4. But it has obvious slowdown when 8, because of the decreasing of space tile size.

Fig.5.6 Performance with time tile size

6 Related Works

Many-core architectures have been considered by both academia and industry. Besides Larrabee and Cyclops-64 mentioned earlier, [30][31][32][33][34] also presented many-core researches, including IBM Cell, Merrimac and nVidia GeForce. Godson-T is a many-core chip with new features. There are many researches on it, including performance optimization [17,18,19] and architecture design [16].

On many-core architecture, performance tuning for specific application is a research focus, including matrix-multiplication [5], irregular computation [19], LU decomposition [35], FFT [36]. There experiences give us important guidelines during our performance tuning, e.g. using SPM, loop tiling, register tiling, instruction scheduling, etc. We adjust these optimizations aiming at the new target (Godson-T) and application (stencil), with new hardware and application features into consideration.

Stencil is important for many scientific/engineering applications, as mentioned earlier. Stencil optimization has been researched on single-core processors [7, 11, 12, 21] for many years to achieve good data locality. And it is also a research focus on multi/many-cores in recent years [6, 8, 9, 10, 13, 14, 22], in order to increase data locality, obtain good parallelism, balance workload and reduce memory traffic. We extend the critical issues with globally accessed SPM and fine-grain synchronization exploration.

Related works most relevant to this paper have been discussed in earlier section (Section 3.2), and here we will not repeat.

7 Conclusion & Future work

Our results demonstrate that globally-accessed SPM and fine-grain memory

based synchronization make great contributions to stencil code running on Godson-T. We design and implement a number of many-core based optimizations for stencil code. Our experimental study show good performance improvements in both execution time speedups and scalability, validated the value of globally accessed SPM and fine-grain synchronization mechanism (full-empty bits) under the Godson-T.

Our work also provides some useful guidelines for future compiler technology of many-core architecture. First, our optimization methods can be incorporated in future many-core compiler/tools to explore the performance advantages. Second, Traditional optimizations still play important roles on many-core. Third, the parameters for compiler optimizations can be statically determined.

For 1-D Jacobi, the surface-to-volume ratio is small, and cost of load/store of the boundary points does not play a critical role for the overall performance. But for multiple-dimensional problem, as the surface-to-volume ratio increases, we should pay more attention to the data layout on SPM, e.g. if each core reads boundary data from its preceding core as we did for 1-D Jacobi, cores with index (x,0) have to pay longer latency than others due to the longer distance to its preceding core(x-1, 7), so that these cores become the slower points of the chip. We will consider this problem in the future.

References

[1] W. J. Dally. Computer architecture in the many-core era. In keynote at the 24th Intl. Conf. on Comput. Design, Oct 1, 2006. San Jose, CA, USA.

[2] S. Y. Borkar, H. Mulder, P. Dubey, S. S. Pawlowski, K. C. Kahn, J. R. Rattner, and D. J. Kuck. Platform 2015: Intel processor and platform evolution for the next decade, 2005.

[3] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Pradeep Dubey1, Stephen Junkins1, A.L., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T. & Hanrahan, P. Larrabee: A Many-Core x86 Architecture for Visual Computing. ACM Transactions on Graphics Vol. 27.

[4] Weirong Zhu, Vugranam C. Sreedhar, Ziang Hu, Guang R. Gao: Synchronization state buffer: supporting efficient fine-grain synchronization on many-core architectures. ISCA 2007: 35-45, San Diego,California,USA

[5] Ziang Hu, Juan del Cuvillo, Weirong Zhu, Guang R. Gao: Optimization of Dense Matrix Multiplication on IBM Cyclops-64: Challenges and Experiences. Euro-Par 2006: 134-144, Dresden, Germany.

[6] S. Krishnamoorthy, M. Baskaran, U. Bondhugula, J. Ramanujam, A. Rountev, P. Sadayappan, "Effective Automatic Parallelization of Stencil Computations," in

Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation, June 2007, San Diego, California, USA.

[7] M. Frigo and V. Strumpen. The memory behavior of cache oblivious stencil computations. Journal of Supercomputing, 2006.

[8] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick. Implicit and explicit optimizations for stencil computations. In Proceedings of MSPC '06, pages 51–60, 2006, San Jose, California, USA.

[9] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, K. Yelick, Stencil Computation Optimization and Auto-tuning on State-of-the-Art Multicore Architectures, SC2008, Austin, Texas, USA.

[10] L. Renganarayanan, M. Harthikote-Matha, R. Dewri, and S. V. Rajopadhye. Towards optimal multi-level tiling for stencil computations. In IPDPS, pages 1.10. IEEE, March 2007, Long Beach, California, USA.

[11] J. McCalpin and D. Wonnacott. Time skewing: A value-based approach to optimizing for memory locality. Technical Report DCS-TR-379, DCS, Rugers University, 1999.

[12] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. In Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation, 1999, Atlanta, Georgia, USA.

[13] D. Wonnacott. Using time skewing to eliminate idle time due to memory bandwidth and network limitations. In IPDPS:Interational Conference on Parallel and Distributed Computing Systems, 2000, Cancun, Mexico.

[14] M. Baskaran, U. Bondhugula, S. Krishnamoorthy, J. Ramanujam, A. Rountev and P. Sadayappan, "Automatic Data Movement and Computation Mapping for Multi-level Parallel Architectures with Explicitly Managed Memories," in Proc. 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, (PPoPP 2008), Salt Lake City, Utah, USA.

[15] K. Datta, S. Kamil, S. Williams, L. Oliker, J. Shalf, K. Yelick, "Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors", SIAM Review, 2008.

[16] He Huang, Nan Yuan, et al, architecture supported synchronization-based cache coherence protocol for many-core processors, CMP-MSI, 2008, Beijing, China.

[17] Xiaochun Ye, Van Hoa Nguyen, Dominique Lavenier, Dongrui Fan, "Efficient Parallelization of a Protein Sequence Comparison Algorithm on Manycore Architecture," pdcat,pp.167-170, 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2008, Dunedin, Otago, New Zealand.

[18] Guoping Long, Dongrui Fan, et al: A Performance Model of Dense Matrix Operations on Many-Core Architectures. Euro-Par 2008: 120-129, Las Palmas de Gran Canaria, Spain.

[19] Guangming Tan, Dongrui Fan, Junchao Zhang, Andrew Russo, Guang R. Gao: Experience on optimizing irregular computation for memory hierarchy in manycore architecture. PPOPP 2008: 279-280, Salt Lake City, Utah, USA.

[20] R. Alverson, D. Callahan, et.al. The Tera compute system, SIGARCH Comput. Archit. News, 18(3b):1-6, 1990.

[21] Michael E. Wolf and Monica S. Lam, A Data Locality Optimizing Algorithm, ACM SIGPLAN Conf. Progr. Lang. Design and Implementation (1991), Toronto, Ontario, Canada.

[22] Chau-Wen Tseng: Compiler Optimizations for Eliminating Barrier Synchronization. PPOPP 1995: 144-155, Santa Barbara, California, USA.

[23] Juha Haataja, Ville Savolainen, Cray T3E User's Guide, (Center for Scientific Computing, Finland, 1997).

[24] B. Smith. The architecture of HEP. In J. S. Kowalik, editor, Parallel MIMD Computation: HEP Supercomputer and Its Applications, Scientific Computation Series, pages 41–55. MIT Press, Cambridge, MA, 1985.

[25] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The Tera computer system. SIGARCH Comput. Archit. News, 18(3b):1–6, 1990.

[26] W. J. Dally and et. al. The message-driven processor. IEEE Micro., 12(2):23–39, 1992, Portland, Oregon, USA.

[27] D. Kranz, B. H. Lim, and A. Agarwal. Low-cost support for fine-grain synchronization in multiprocessors. Technical Report MIT/LCS/TM-470, 1992.

[28] S. W. Keckler, W. J. Dally, D. Maskit, N. P. Carter, A. Chang, and W. S. Lee. Exploiting fine-grain thread level parallelism on the MIT multi-ALU processor. In the Procs. of 25th Intl. Symp. on Computer Architecture, 1998, Barcelona, Spain.

[29] Cray MTA-2 System.

[30] John Montrym, Henry Moreton. The GeForce 6800. IEEE Micro, Volume 25 Issue 2, March 2005.

[31] P. Hofstee. Power Efficient Architecture and the Cell Processor. Invited Paper and Keynote Speech. HPCA-11, February 2005, San Francisco, CA, USA.

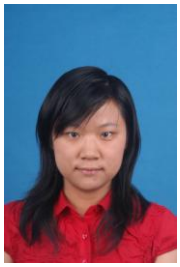
[32] Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker,W.L., Shalf, J.,Williams, S.W.,Yelick, K.A.: The Landscape of Parallel Computing Research: A View from Berkeley

[33] Vangal, S., Howard, J., Ruhl, G., Dighe, S.,Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T., Jain, S., Venkataraman, S., Hoskote, Y., Borkar, N.: An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In: Proceedings of IEEE International Solid-State Circuits Conference, February 11-15 (2007), San Francisco, CA, USA.

[34] Dally,W.J., Labonte, F., Das, A., Hanrahan, P., Ahn, J.H., Gummaraju, J., Erez, M., Jayasena, N., Buck, I., Knight, T.J., Kapasi, U.J.: Merrimac: Supercomputing with Streams. In: Proceedings of the Supercomputer Conference, November 15-21 (2003), Phoenix, Arizona, USA.

[35] Ioannis E. Venetis and Guang R. Gao, Mapping the LU Decomposition on a Many Core Architecture: Challenges and Solutions, ACM International Conference on Computing Frontiers (CF2009), Ischia, Italy. May 18-20, 2009, Ischia, Italy.

[36] Liping Xue, Long Chen, Ziang Hu, and Guang R Gao, CAPSL Technical Memo 81: Performance Tuning of the Fast Fourier Transform on a Multicore Architecture



Huimin Cui is a PhD candidate in the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, CAS. Her research interests are in the compiler, runtime system and binary translation areas. She received her bachelor and Master degrees in computer science from Tsinghua University at 2001 and 2004 respectively.

Lei Wang was born in 1976. She received her B.E. degree from Beijing Institute of Technology in 1999 and M.S. degree from Beijing Institute of Technology in 2002. She is currently an assistant professor of the Key

Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences. Her research interests include compiler and runtime system.

Dongrui Fan graduated from the Department of Mathematical Science at Beijing Jiaotong University with a bachelor degree at 2000, and he achieved Ph.D. degree from Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) at 2005. Now, he is an associate professor in ICT, IEEE member and CCF member. He worked together with the members of AMS(Advanced Micro-System) research group and designed the new processing models--Godson-X and Godson-T. Currently, His research interest focuses on many-core system, including the design of microarchitecture, parallel processing, and runtime system.

Xiaobing Feng was born in 1969. He received his B.E. degree from Tianjin University in 1992, M.S. degree from Peking University in 1996 and Ph. D. degree from the Institute of Computing Technology, Chinese Academe of Sciences. He is currently a professor of the Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences. His research interests include program analysis, compiler and tools.