Tsinghua Science and Technology

Volume 21 | Issue 5 Article 11

2016

HW/SW Co-optimization for Stencil Computation: Beginning with a Customizable Core

Yanhua Li
Department of Computer Science, Tsinghua University, Beijing 100084, China.

Youhui Zhang
Department of Computer Science, Tsinghua University, Beijing 100084, China.

Weiming Zheng
Department of Computer Science, Tsinghua University, Beijing 100084, China.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 11/11 pp570-580
Volume 21, Number 5, October 2016

HW/SW Co-optimization for Stencil Computation: Beginning with a Customizable Core

Yanhua Li, Youhui Zhang*, and Weiming Zheng

Abstract: Energy efficiency is one of the most important issues for High Performance Computing (HPC) today. Heterogeneous HPC platform with some energy-efficient customizable cores (as application-specific accelerators) is believed as one of the promising solutions to meet ever-increasing computing needs and to overcome power density limitations. In this paper, we focus on using customizable processor cores to optimize the typical stencil computations — the kernel of many high-performance applications. We develop a series of effective software/hardware co-optimization strategies to exploit the instruction-level and memory-computation parallelism, as well as to decrease the energy consumption. These optimizations include loop tiling, prefetching, cache customization, Single Instruction Multiple Data (SIMD), and Direct Memory Access (DMA), as well as necessary ISA extensions. Detailed tests of power-efficiency are given to evaluate the effect of all these optimizations comprehensively. The results are impressive: the combination of these optimizations has improved the application performance by 341% while the energy consumption has been decreased by 35%; a preliminary comparison with X86, GPU, and FPGA platforms also showed that the design could achieve an order of magnitude higher performance efficiency. We believe this work can help understand sources of inefficiency in general-purpose chips and can be used as a beginning to customize an energy efficient CMP for further improvement.

Key words: energy efficiency; customizable processor; stencil computation; software and hardware co-optimization

1 Introduction

Power consumption has become one of the most essential issues in High Performance Computing (HPC) system designs for a number of reasons including cost, reliability, energy conservation, and environmental impact.

The E3 report[1], which proposed a new initiative, had disclaimed that an Exaflops system will require more than 200 megawatts (MW) of sustained power consumption if existing technology is simply extrapolated into the future.

Now, much research has proved that efficient designs must be specific to application and/or algorithm classes. They usually employed some alternative approaches (instead of general-purpose processors) to construct the high-performance computing systems, including FPGA, embedded processors, and system-on-chip. For example, the BlueGene and SiCortex[2] supercomputers are based on embedded processor cores that are more typically seen in automobiles, cellphones, and so on; FPGA has been also used as application-accelerators in computing systems more and more[3–5].

Among these alternative approaches, customizable processors can be regarded as a promising method. Green Flash[6, 7] is such an example in the supercomputing field. The Green Flash system employs power-efficient cores specifically tailored to meet the requirements of the ultra-scale climate code. They intend to achieve 100 times the computational efficiency and 100 times the capability of the mainstream HPC

• Yanhua Li, Youhui Zhang, and Weiming Zheng are with Department of Computer Science, Tsinghua University, Beijing 100084, China. E-mail: liyh09@mails.tsinghua.edu.cn; zyh02@tsinghua.edu.cn; zwm-dcs@mail.tinghua.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2015-09-07; accepted: 2016-02-03

approach. Another example is Anton[8]. Anton is a special-purpose supercomputer designed for Molecular Dynamics (MD) simulations, which uses the ASIC technology and customizable processors together to drastically increase the speed of MD calculations, about two orders of magnitude beyond the previous state of the art. In other fields like H.264 encoding, customizable processors can also achieve very good results. As Hameed et al.[9] disclaimed, it was feasible to achieve ASIC energy levels in a customized processor by creating customized hardware that easily fit inside a processor framework; extending a processor looks like the correct approach to get a good balance between power efficiency and programming flexibility.

Enlightened by these ideas, this paper is focused on how to use the customizable processor cores to optimize the stencil (nearest-neighbour) computations—a class of algorithms at the heart of most scientific calculations involving structured grids, including both implicit and explicit Partial Differential Equation (PDE) solvers. The design philosophy is just opposite to that of general-purpose processors: the latter considers much on the backward-compatibility and the wide range of application-adaptability. Thus for a concrete class of applications of algorithms, general-purpose processor usually cannot achieve a fairly high power-efficiency. In contrast, we enhance a single and very simple core for stencil computations. We try to exploit the instruction-level and computation-memory parallelism, as well as to improve the energy efficiency by employing those needed architecture features combined with software optimization, including data prefetch, on-chip memory, cache-strategy customization, Direct Memory Access (DMA), and Single Instruction Multiple Data (SIMD) methods.

We believe this method can help us to find out the most power-efficient way for the target computing. In some sense, it will also help us to understand the sources of inefficiency in general-purpose chips.

In summary, this paper presents the following contributions:

- We illustrate how to adapt a simple core (as the accelerator) to stencil computation. Besides the common software optimizations (like array padding, loop tiling, and so on), a wide variety of hardware customizations have been used, including data prefetch, on-chip memory for temporary storage, online adjusting the cache strategy to reduce memory traffic, DMA for the overlap of computation and memory-IO, and SIMD to explore the ILP.
- Based on the solution provided by Tensilica[10], we have evaluated each optimization step from the aspects of core performance, power consumption, and so on, which helps us to construct a comprehensive assessment of the optimization strategies.
- The result of core customization provides a very desirable performance/energy choice over general purpose processors and even FPGA for constructing energy-efficient heterogeneous HPC platforms.

Now only one core's architecture space with the single-precision floating point computation has been explored. In the next step, we will design the corresponding CMP chip, containing tens of cores and a customized network-on-chip, to improve the whole efficiency and performance further.

This paper is organized as follows: Section 2 discusses related work, Section 3 introduces stencil computation and the customization flow, Section 4 describes the customization design in detail, Section 5 is evaluation and analysis, and Section 6 gives the conclusions and future work.

2 Related Work

From the system viewpoint, customizable design is a hardware/software co-design process: system designers and application scientists cooperate with each other in the early stages to tailor the system architecture to the application resource requirements. Cong et al.[11] just carried out the research on a general, customizable platform that can be customized to a wide-range of applications.

CPUs tailored to meet the requirements of high performance computing applications have been widely used. Besides the embedded processor cores (PowerPC A2) of the Blue/Gene supercomputers, the K computer[12] also customized its SPARC64 VIIIfx cores for high performance applications, including the SIMD extension, a larger floating point register file, etc.

Green Flash[6, 7, 13] claimed that it has proposed a new approach to scientific computing that chooses the science target first and then designs systems for applications (rather than the reverse). Especially the Green Flash system tailored Xtensa processor-cores for the parallel models. Its processor design is focused on how to build the CMP- and higher-level structures. In

contrast, we focus on the single core's design.

Anton[8] is designed and built specially for MD simulations of proteins and other biological macromolecules. Besides 32 deeply pipelined ASIC modules, each Anton processing node contains four Tensilica cores and eight specialized but programmable SIMD cores (called geometry cores).

Customized CPUs can also match an ASIC solution's performance within 3× of its energy and within comparable area for H.264 encoding. Hameed et al.[9] customized Xtensa cores with three broad strategies: (1) Very Long Instruction Word (VLIW) and SIMD techniques, (2) merging frequently-used sequential operations into a single instruction, and (3) the creation application-specific data storage with fused functional units.

Some research has been carried out to generate customization automatically. Atasu et al.[14] presented Fast Instruction SynthHesis (FISH), a system that supports automatic generation of custom instruction processors from high-level application descriptions to enable fast design space exploration. Grad and Plessl[15] studied the feasibility of instruction set specialization for reconfigurable ASICs at runtime; they proposed effective ways of pruning the design space which can reduce the runtime of instruction set extension algorithms by two orders of magnitude.

Datta et al.[16] developed a number of effective software optimization strategies of stencil computation for a broad set of multi-core architectures in the current HPC literature, including X86/UltraSparc CPUs, GPU, and CELL. But no customizable processors had been touched. Membarth et al.[17] presented a domain-specific approach to automatically generate code (low-level CUDA and OpenCL) tailored to different processor types, instead of writing hand-tuned code for GPU accelerators.

3 Stencil Computation Analysis and Customization Platform

3.1 Stencil computation analysis

Stencil computation is widely used in scientific computing, engineering applications, image processing, and cellular automata. Applications are often performing nearest neighbour computations called stencils. In a stencil operation, each point in a grid is updated with weighted contributions from a subset of its neighbours. These operations are then used to build solvers that range from simple Jacobi iterations to complex multi-grid and adaptive mesh refinement methods[18].

As described in Ref. [19], the stencil computation can be presented by Eq. (1) over a 3-D rectangular domain:

$$C(i,j,k)=\sum_m A_m\cdot B(i\pm I_m, j\pm J_m, k\pm K_m)+\sum_l A_l(i,j,k)\cdot B(i\pm I_l, j\pm J_l, k\pm K_l) \quad (1)$$

The center point and some neighboring points in the input grid (Array B) are weighted by either scalar constants (A_m) or elements in grid variables ($A_l(i, j, k)$) at the same location as the output. Offsets ($I_m/J_m/K_m$ and $I_l/J_l/K_l$) that constrain how the input data is accessed are all constant. We call their maxima the halo margins of three dimensions. Then, a stencil calculation is called an N-point stencil where N is the total number of input points used to calculate one output point.

In this paper we select one of computation kernels of Helmholtz Solver as the example, nearly expressed as triply nested loops. This nineteen-point stencil (shown in Fig. 1) performs an out-of-place iteration; the halo value is 1 for all three dimensions. Values are weighted by elements in grid variables, A, which apparently increases the data-transmission pressure and lacks the temporal locality. For each grid point, this stencil will execute 37 floating point operations and 39 memory references.

The data structures of stencil calculations are typically much larger than the capacity of data caches. In addition, the amount of data reuse is limited to the number of points in a stencil. As a result, these computations generally achieve a low fraction of theoretical peak performance, since

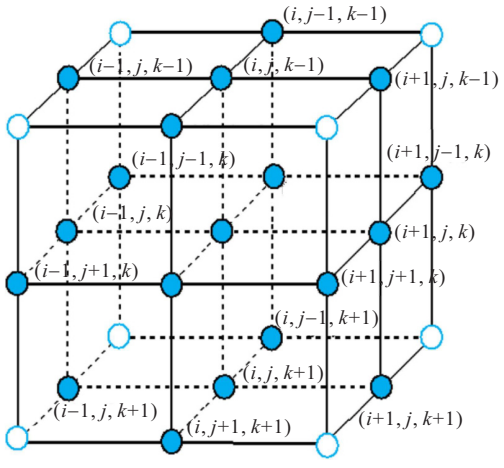

Fig. 1 The 19-point stencil computation.

data from main memory cannot be transferred fast enough to avoid stalling the computational units on modern microprocessors. Reorganizing these stencil calculations to take full advantage of memory hierarchies has been the subject of much investigation over the years. A study of stencil optimization[20] on (single-core) cache-based platforms found that tiling optimizations were primarily effective when the problem size exceeded the on-chip cache's capacity.

3.2 Customization platform

Customizable processor is one of the key components for this work. Such a fully featured processor usually consists of a base processor design and a design-tool environment that permits significant adaptation of that base processor by allowing system designers to change major processor functions. Especially the instruction set can be extended by the application developer.

Tensilica Xtensa is a typical instance, which provides a base design that the designer can extend with custom instructions and data-path units[12]. Manually creating ISA extensions gives larger gains: Tensilica reports speedups of 40× to 300× for kernels such as FFT, AES, and DES encryption.

We use the Xtensa Xplorer IDE as the design/evaluation tool and use the LX4 core as the starting point of enhancement. The IDE contains a cycle-accurate core simulator with energy estimator, which can be automatically recreated according to the latest core-configuration. Thus, we can get the running cycles/frequency, power consumption, and the chip area under a given CMOS process.

In this paper, the base LX4 core has the following features: 32-bit architecture with a compact 24-bit instruction set including 80 RISC instructions; one seven-stage/single-issue pipeline; 32 general-purpose 32-bit registers; a single-precision floating point processing unit; the MMU; one L1 i-cache and one L1 d-cache (both are direct-mapped and the size is 4 KB). It can achieve 1 GHz on 45 nm GS process technology. LX4 supports the PIF interface to connect the main memory, which completes one data access-transaction in 1–60 cycles on average and the interface width is up to 16 bytes. Moreover, two load/store units can be used synchronously to increase the throughput further.

4 HW/SW Co-optimization Strategies and Implementation

According to the characteristics of the stencil computation, both the memory accessing and computing pattern, some common architectural optimization features as well as application-specific hardware are applied to the customizable processor one by one. In the customization flow, some optimizing features are used for the ILP exploration, some for bandwidth optimizations and others for the overlap of memory-IO and computation. Other than these hardware customizations, software optimizations (like array padding and loop tiling) combined with cache adaptation were also applied.

HW/SW co-optimization strategies for stencil computation are as follows:

- Software optimizations (array padding and loop tiling) combined with cache adaption;
- Floating-point SIMD instruction(s);
- DMA combined with memory on-chip solutions;
- Bandwidth optimizations, including data prefetching and adjusting the cache attribute to prevent filling the overwritten regions.

This work tries to apply them one by one to the customizable processor and evaluate their results respectively in the subsequent sections. Each will be evaluated with a variety of metrics, including the running performance, energy-consumption, chip-area, cache-miss rate, memory accesses, and so on.

4.1 Loop tiling and array padding

Tiling and array padding are common software optimizations. The novelty here is: Owing to the features of customizable processor, the cache's configuration can be customized, too. Therefore, we can co-design hardware/software together to balance the performance, the energy consumption, and the chip area.

In detail, the array padding transformation sets a dimension in an array to a new size in order to reduce the number of cache-set conflicts; tiling for locality requires grouping points in an iteration space into smaller blocks (tiles) allowing reuse in multiple directions when the block fits in a faster memory (L1 cache in this work). Both are directly dependent on the d-cache's configurations.

Specific to stencil computations, as mentioned in Ref. [21], to carefully select tile dimensions is important to avoid cache conflicts. Because there are quite a few researches that have explored the tiling optimizations for stencil, here we only present the brief optimization principles.

Without loss of generality, we take a typical 3-D stencil, 3-D Jacobi iteration solver (shown in Fig. 2) as the example.

Generally, the original code can be transformed into Fig. 3, which has tiled the three loops and permuted kk, jj, and ii to the outermost level. It means that the cache should hold a $(\mathrm{ti}+2)\times(\mathrm{tj}+2)\times(\mathrm{tk}+2)$ sub-array and decrease the conflict-number to the least. Therefore, we have to compute the number of cache misses as the number of distinct cache lines accessed in each ti×tj×tk block of iterations. Euc3d[21, 22] is such an algorithm, which initially computes several non-conflicting array tiles and then selects from among them the tile minimizing the cost.

Euc3d takes the cache size and line length as the input; in our design the cache's configuration can be customized, too. Therefore, for a given cache-configuration, we use the Euc3d to compute the optimized tile(s) and complete the system evaluation on the processor simulator to get more precise results, like running time, energy consumption, and so on. Then, we adjust the cache-configuration, repeat this work flow, and try to find a comprehensively optimized design, which is one of the unique characteristics of processor-customization. With the help of Pluto[23], an automatic locality-optimization tool that can achieve the source-to-source transformation, we have tried different strategies of loop tiling and data padding on the core simulator, while the cache configuration can

```
do k=2, N−1
do j=2, N−1
do i=2, N−1
A(I, j, k)=
C×(B( i−1, j, k)+B(i+1, j, k)+
B( i, j−1, k)+B(i, j+1, k)+
B( i, j, k−1)+B(i, j, k+1))
```

Fig. 2 FORTRAN code of a typical 3-D Jacobi iteration.

```
do kk=2, N−1, tk
do jj=2, N−1, tj
do ii=2, N−1, ti
do k=kk, min(kk+tk−1, N−1)
do j=jj, min(jj+tj−1, N−1)
do i=ii, min(ii+ti−1, N−1)
A(I, j, k)=C×(B(i−1, j, k)+
B(i+1, j, k)+B(i, j−1, k)+B(i, j+1, k)+
B(i, j, k−1)+B(i, j, k+1))
```

Fig. 3 The converted equivalent code of a 3-D Jacobi iteration.

be adjusted. Then, we found out the best strategy and the corresponding cache configurations: one 16 KB, two-way set-associative write-back d-cache with the 256 bytes cache-line.

From the viewpoint of software, the best tiling strategy is to calculate data tiles one by one along the i, j, and k directions sequentially and the tile size is 64×3×3. Of course, more capacities or higher set-association degrees will lead to better performance. However, we found that the performance improvement is limited while the additional chip area is relatively large. Therefore, we select this compromise configuration, which is called Case Tiling in the following subsections.

4.2 SIMD

As we know, the ability of floating point processing is the shortcoming of normal customizable cores: they usually own a single floating point processing unit and no SIMD is supported. Therefore, we have to enhance the corresponding capability to adapt to high-performance computing.

For stencil, the good news is that it may be very efficient to employ SIMD instructions to speed up computation because there is almost no data-dependence in the innermost loop. The bad news is that the real challenge lies in the performance of data access. We intend to use the DMA engine to transfer data from the external memory into the memory on-chip directly to overlap the communication with computation. It works like the Cell processor that transfers data to and from the main memory and local memories. However, most modern processors lack this feature.

To enhance the floating-point processing capability, we employ the ISA extension technique of customizable CPUs. In detail, we implement the add and multiply operations of floating points using the integer operations. Because the SIMD technique of integer operations is supported inherently by LX4, we further implement the SIMD version of add and multiply operations of floating points.

The detailed design flow is follows.

Step 1 According to the ANSI/IEEE Std 754-1985 specification, we implement the add and multiply operations (referred to as float-add and float-mul in the following contents) of two single-precision floating-point values using the TIE (Tensilica Instruction Extension) language.

Step 2 We introduce one 288-bit register and three

96-bit registers (referred to as vec and mat1 3 in the following contents) that are used to occupy up to 9 consecutive values of Array A and three groups of 3 consecutive values of Array B, respectively.

Step 3 Based on the instruction-fusion and SIMD mechanisms of TIE, we design the final instruction and operation unit, which can complete up to nine "add and multiply" operations in one instruction. For example, all computations in Eq. (2) can be completed by one instruction; the TIE pseudo code is presented in Fig. 4.

$$\begin{aligned}\text{TEMP\_SUM} = {} & A(1,i,j,k) \times B(i,j,k) + \\ & A(2,i,j,k) \times B(i-1,j,k) + \\ & A(3,i,j,k) \times B(i+1,j,k) + \\ & A(4,i,j,k) \times B(i,j-1,k) + \\ & A(5,i,j,k) \times B(i,j+1,k) + \\ & A(6,i,j,k) \times B(i+1,j+1,k) + \\ & A(7,i,j,k) \times B(i+1,j-1,k) + \\ & A(8,i,j,k) \times B(i-1,j-1,k) + \\ & A(9,i,j,k) \times B(i-1,j+1,k) \end{aligned} \quad (2)$$

Of course, the new registers and instruction can be reused for other SIMD computations.

Now, the new scalar MADD instruction is able to support the computation of up to 9 pairs of floating point objects in parallel. Tests show that the new instruction can complete the computation of the innermost loop in 30 cycles (if the cache is perfect), while the original FPU spends about 80 cycles (the cache is perfect).

4.3 DMA combined with memory on-chip solutions

General speaking, DMA allows certain hardware-module to access system memory independently of the CPU. Owing to the DMA, the core can do other operations as the data transfer is in progress, which is especially meaningful if the data amount is large and/or the transfer is relatively slow.

We intend to use the DMA engine to transfer data from the external memory into the memory on-chip directly to overlap the communication with computation. It works like the Cell processor that transfers data to and from the main memory and local memories. However, most modern processors lack this feature.

The Xtensa processor can contain up to 128 KB local SRAM and its access latency is just one cycle. We allocated a double-buffer in the local memory for the DMA transfer. It also supports in-bound DMA option, which means that other cores or DMA engines can transfer data into the local memories directly in parallel with the processor pipeline. The software can deal with the completion of DMA operations through polling mode or interrupt mode.

In our implementation, a simple 7-stage Xtensa LX processor is used as the DMA engine and the system sketch map is presented in Fig. 5. These two cores are connected with two TIE ports: one is used to export the core's DMA request to the controller; the other feeds back the operation results.

We focus on Array A, whose transmission pressure is the largest; other data is still accessed through the d-cache. The whole computation procedure can be described as follows:

(1) Initialize the engine;

(2) DMA the first data-segment of Array A into Buffer One;

(3) Wait for the completion (in polling mode);

(4) DMA the follow-up data into Buffer Two;

(5) Execute the floating-point-computations on Buffer One (other data accesses are handled as

```
operation Nine_SIMD_MulAdd{out AR acc, in vec32x9
vec, in vec32x3 matl, in vec32x3 mat2, in vec32x3 mat3}
{
wire  [31:0]p10 = floatmul (vec [287:256], mat2 [63:32]);
wire  [31:0]p11 = floatmul (vec [255:224], mat2 [95:64]);
wire  [31:0]p12 = floatmul (vec [223:192], mat2 [31:0]);
wire  [31:0]t10 = Fadd3(p10, pll, p12);
wire  [31:0]p20 = floatmul (vec [191:160], matt [63:32]);
wire  [31:0]p21 = floatmul (vec [159:128], mat3 [ 63:32]);
wire  [31:0]p22 = floatmul (vec [127:96], mat3 [31:0]);
wire  [31:0]t20 = Fadd3(p20, p21, p22);
wire  [31:0]p30 = floatmul(vec[95:64], matl[31:0]);
wire  [31:0]p31 = floatmul(vec[63:32], matl[95:64]);
wire  [31:0]p32 = floatmul(vec[31:0], mat2[95:64]);
wire  [31:0]t30 = Fadd3(p30, p31, p32);
assign acc = Fadd3(t10, t20, t30) ;
}
```

Fig. 4 Pseudo code of the new SIMD instruction.

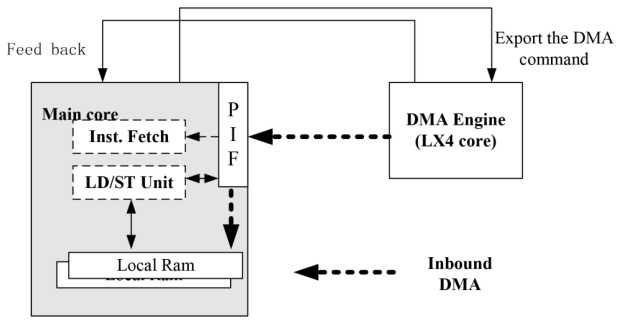

Fig. 5 The inbound DMA.

normal)

(6) DMA the follow-up data into Buffer One;

(7) Execute the computations on Buffer Two;

(8) Go to Step (4) till the end.

Thus DMA operations are decoupled from execution.

Moreover, the introduction of DMA (and local memories) can decrease the pressure on the d-cache remarkably because only the data of Arrays *B* and *C* will be cached. Then, we have gotten another best cache configuration: one 8 KB, two-way set-associative d-cache with the 64 bytes cache-line. For this configuration, the performance is almost the same as the larger cache, while the area cache and energy consumption both decrease.

4.4 Bandwidth optimizations

To reduce the required bandwidth, we employ two types of optimization: (1) HW/SW pre-fetch mechanisms; and (2) adjusting the cache strategy during the run-time to avoid the stencil-specifically unnecessary data accesses.

For hardware pre-fetch, the Xtensa processor offers such an option. Internally, the pre-fetch logic has a miss history table which keeps track of the data cache misses that have occurred; and the pre-fetch buffers that keep track of the cache line addresses will be pre-fetched. There are a configurable number of pre-fetch buffers (8 or 16). It also supplies some pre-fetch instructions, DPFR, DPFRO, DPFW, and DPFWO; all pre-fetch a single cache line into the pre-fetch buffer.

In addition, Array *C* of the target computation will be overwritten; thus the attribute of allocate-on-write-miss is a burden for it. Then we are faced with a trade-off because the usual attribute of write-back-and-allocate-on-write-miss has been proved better for common case. Fortunately, LX4 processors support data-region control operations that can set cache attributes of a given region of memory, so that we can set the cache attributes of Array *C* as write-back-but-no-allocate-on-write-miss while keeping the remaining part as the normal (write-back-and-allocate-on-write-miss). In detail, the processor modifies the corresponding attributes in the Memory Management Unit (MMU).

5 Evaluation and Analysis

We conduct the experimental environment based on the IDE provided by Tensilica. We have carried out evaluations for each above-mentioned optimization step carried out and examined from the aspects of core performance, power consumption, and chip area. The base array scale for stencil computation test is set as 512×512×3. We also compare the design with other platforms, including X86, GPU, and FPGA.

5.1 Performance, energy, and area of the proposed optimization strategies

Figure 6 shows the performance and energy consumption of each step of the proposed co-optimization. The combination of these optimization steps has improved the application performance by 341% while the energy consumption has been decreased by 35%. The detailed analysis of each optimization strategy is as follows.

(1) Software optimization (Tiling and padding combined with cache customization)

At first, we test the naïve version on the preliminary core. And then, we optimize the source code as described in Section 4.1 and adjust the configurations of data cache at the same time to find out the best combination.

We found that one 16 KB, two-way set-associative d-cache with the 256 bytes cache-line (which is called Case Tiling and used as the reference in the following comparisons) is the most optimized: its performance is almost 2.4 times higher than the naïve version (16 KB, 2-way set-associative, 64 bytes cache-line) while the energy consumption is only about 55% (the typical case), which is presented in Table 1.

For Case Tiling, the miss rate of d-cache is about 1/19 of the naïve version (1.4% vs. 27%). Of course, more capacities or higher set-association degrees will lead to limited performance improvement while they have caused more chip resource or/and power consumption.

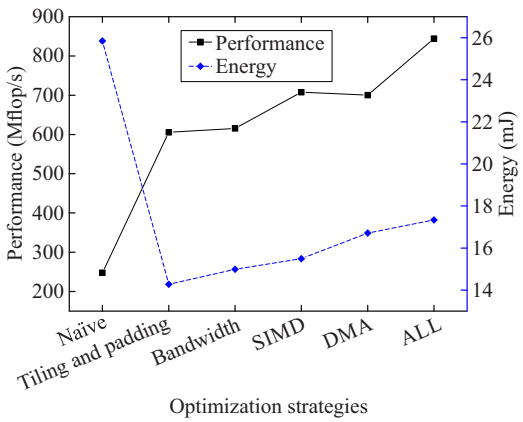

Fig. 6 Performance and energy consumption of each optimization strategy.

Table 1 Results of tiling and padding.

	Frequency (GHz)	Number of cycles ($\times 10^9$)	Energy (mJ)
Case Naïve	1.399	0.159	25.851
Case Tiling	1.398	0.065	14.277

Therefore, we select this configuration finally.

(2) Bandwidth optimization (Pre-fetch and no-allocate cache strategy)

Our tests show that the hardware pre-fetch itself hardly brings any improvement; its software counterpart can get very limited performance promotion.

We believe the reason is: the cache line is fairly long (256 bytes), therefore the data used in the near future has been loaded implicitly; moreover, the tiling technology has utilized the locality fully.

To set the cache attributes of Array C as write-back-but-no-allocate-on-store improves the performance by about 1.6% and increases the energy consumption by 2% compared to Case Tiling. In detail, test results show that this optimization can reduce the memory throughput by about 3.7%, which does correspond to the improvement.

(3) SIMD

The ILP optimization has improved the real application performance by about 15% while increasing the energy consumption by 8%. Because this application is mainly memory-bound, the computation speedup cannot improve the application performance as much as in the ideal situation (Section 4).

Moreover, the number of data-load operations is only 76.5% of the reference case. We believe that the three customized registers have increased the effective storage capacity on-chip; therefore some memory reads have been avoided.

(4) DMA

As described in Section 4, we load the data into the local RAM in parallel with the computation. In detail, two banks of 16 KB on-chip RAM are allocated; each is equipped with an independent port to avoid access conflicts. The data size of one DMA transfer-operation is 6 KB.

Because DMA transfers bypass the d-cache, it is reasonable to shrink the cache now: typical-case tests show that DMA can improve performance by 14% and increase the energy by 21% , when the d-cache size is set as 8 KB and the miss rate has been reduced to 0.5%.

In detail, the time overhead of DMA has occupied about 29% of the computation part (including the data load/store operations), which is hidden by computation to an extent degree. On the other hand, the DMA transfer interferes with the normal accesses issued from the d-cache, which partially offsets the overlap.

If the cache-strategy customization and SIMD are both at the same time, the performance increase is 29% while the energy consumption is increased by 17% (typical case). It is the best result we have gotten.

At last, we give die areas of all the above-mentioned configurations (the typical case) in Fig. 7, including areas of each component.

5.2 Energy efficiency

Test results show that, from the viewpoint of performance efficiency, the simple architecture of customizable process is highly efficient: the performance efficiency of Case Tiling is the second highest (the first is Case Bandwidth optimization because it almost introduces no extra overheads). Of course, its absolute performance is not satisfying; thus we use the DMA, SIMD, and other techniques to improve it by about 29% while the energy consumption is only increased by 17%. Thus from the viewpoint of energy efficiency, Case All above is the highest. All these comparisons are presented in Fig. 8.

5.3 Preliminary comparison with X86 and others

Roughly we compare our design with a general-purpose processor, IntelXeonProcessor X5560, which is also produced by the 45 nm CMOS process. The corresponding parameters are listed in Table 2.

X5560 is a four-core processor. Here we simply take one fourth of its nominal power consumption, as well

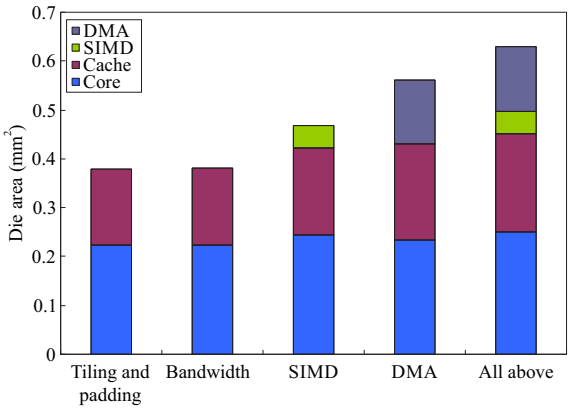

Fig. 7 Die area of different optimization steps.

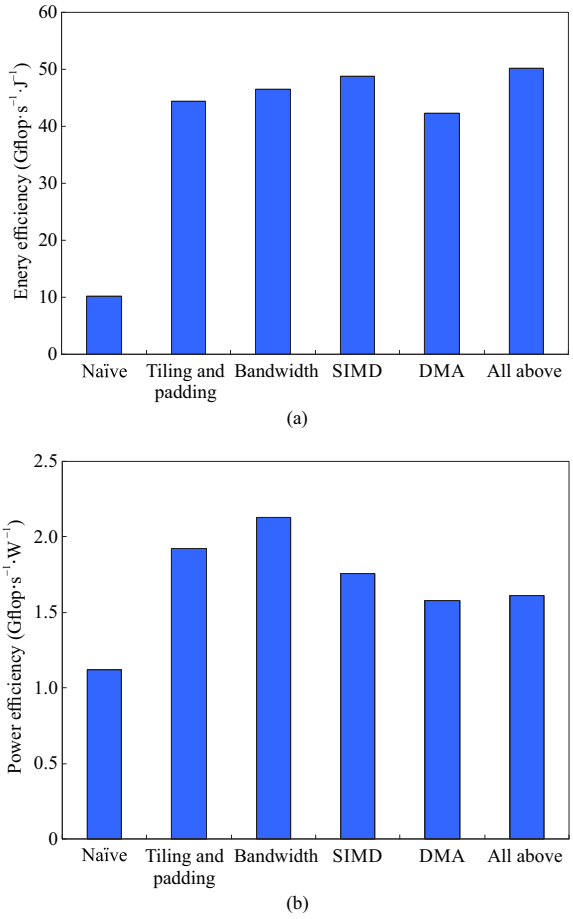

Fig. 8 Power efficiency of all cases.

Table 2 Results of our design over intel X5560.

	Power (W)	Chip area (mm^2)	Frequency (GHz)	Application performance (s)
Our design	0.5	0.914	1.395	33
X5560	24	66	2.8	18

as the chip area. The drawback of this simplification lies in that 1/4 of the shared on-chip 8 MB L3-cache of X86 has been regarded as one part of the core. On the other hand, the whole L3-cache has been used in reality, although only a single core is employed from the aspect of computation. Therefore, we believe this simplification is acceptable.

Then we can see that it can achieve an order of magnitude higher performance efficiency; its application performance is about 50% of that of X86 while the chip area is only 0.91 mm^2, 1/72 of the X86-core area. Of course, we made the simplification here; therefore this comparison is for reference only.

In addition, we compare it with reference designs[23–27] based on GPU or FPGA; the best numbers from them are compared with our performance of the typical case, to provide a fair comparison. Detailed data is listed in Table 3.

From the aspect of performance (throughput), our design is much slower because of its simple structure and limited resource-consumption. But for performance efficiency, it is an order of magnitude higher than GPU and is comparable to FPGA; its programming flexibility is apparently much higher.

6 Conclusion and Future Work

In this paper, we present a systematic approach to enhance a preliminary customizable processor-core with HW/SW co-optimization strategies for stencil computation.

From the results, we can get the following conclusions:

- Customizable cores could play an important role in the power-efficient heterogeneous high performance computing platforms. The customizable core can obtain high power-efficiency and high area-efficiency, owing to keeping the core simple while augmenting necessary architecture features. At the same time, the programming flexibility can be still maintained, which is the unique feature in contrast to the ASIC solution.
- Customizable cores show large HW/SW co-optimization space for the specific applications, like stencil computations. In detail, some data-reuse techniques, like loop tiling and DMA, can efficiently improve performance by increasing the data access speed; specific SIMD extensions can also offset its weakness on the computation capability of floating point data to a certain extent.
- However, not all of the optimization strategies bring performance improvement and energy efficiency. Co-optimization strategies should be carefully used, according to the characteristics of the applications and the actual demands.

Table 3 Results of our design over intel X5560.

		Throughput (Gflop/s)	Efficiency (Mflop·s^{-1}·W^{-1}))
Our design		0.8	1660.0
GPU	Datta et al.[16]	36.0	76.5
	Phillips and Fatic[24]	51.2	n/a
	Yang et al.[25]	64.5	n/a
FPGA	Araya-Polo et al.[26]	35.7	n/a
	Niu et al.[27]	102.8	785.0

Now we are carrying out the development of the chip multi-processor based on the tailored cores, as well as the corresponding parallel algorithms. We intend to embed some hardware supports of the parallel programming modes (including the one and two-side message passing) into the network-on-chip. Especially, the communication work will be offloaded to the specially-designed NoC routers to a great degree, which is beneficial to overlap the communication with computation for higher efficiency.

Acknowledgment

The work was supported by the National High-Tech Research and Development (863) Program of China (No. 2013AA01A215) and the Brain Inspired Computing Research of Tsinghua University (No. 20141080934).

References

[1] Modeling and simulation at the exascale for energy and the environment, Report on the Advanced Scientific Computing Research Town Hall Meetings on Simulation and Modeling at the Exascale for Energy, Ecological Sustainability and Global Security (E3), http://www.sc.doe.gov/ascr/programDocuments/ProgDocs.html, 2007.

[2] S. Tally, New green supercomputer powers up at Purdue, http://www.purdue.edu/uns/x/2008a/080610McCartneySI-Cortex.html, 2008.

[3] F. Xia, Y. Dou, G. Lei, and Y. Tan, FPGA accelerator for protein secondary structure prediction based on the GOR algorithm, *BMC Bioinformatics*, vol. 12, no. S1, p. S5, 2011.

[4] J. Jiang, V. Mirian, K. P. Tang, P. Chow, and Z. Xing, Matrix multiplication based on scalable macro-pipelined FPGA accelerator architecture, in 2009 *International Conference on Reconfigurable Computing and FPGAs*, 2009, pp. 48–53.

[5] L. Liu, O. Neal, B. Chitlur, Q. Wang, C. Alvin, W. Shen, Z. Yu, S. Arthur, M. Ian, G. Joseph, et al., High-performance, energy efficient platforms using in-socket FPGA accelerators, in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2009, pp. 261–264.

[6] M. Wehner, L. Oliker, and J. Shalf, Towards ultra-high resolution models of climate and weather, *International Journal of High Performance Computing Applications*, vol. 22, no. 2, pp. 149–165, 2008.

[7] M. Wehner, L. Oliker, and J. Shalf, Green flash: Designing an energy efficient climate super-computer, in *IEEE International Symposium on Parallel & Distributed Processing, 2009*, 2009.

[8] D. E. Shaw, R. O. Dror, J. K. Salmon, J. K. Salmon, J. P. Grossman, K. M. Mackenzie, J. A. Bank, and E. Chow, Millisecond-scale molecular dynamics simulations on anton, in *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2009, pp. 1–11.

[9] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, and M. Horowitz, Understanding sources of inefficiency in general-purpose chips, in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, vol. 38, no. 3, pp. 37–47, 2010.

[10] Cadence, Tensilica processors, http://ip.cadence.com/knowledgecenter/know-ten, 2016.

[11] J. Cong, V. Sarkar, G. Reinman, and A. Bui, Customizable domain-specific computing, *IEEE Design and Test of Computers*, vol. 28, no. 2, pp. 5–15, 2011.

[12] Fujitsu, K computer, http://www.fujitsu.com/global/about/tech/k/, 2016.

[13] J. Levesque, J. Larkin, M. Foster, J. Glenski, G. Geissler, S. Whalen, and H. Wasserman, Understanding and mitigating multicore performance issues on the AMD opteron architecture, Techincal Report, Lawrence Berkeley National Laboratory, 2007.

[14] K. Atasu, W. Luk, O. Mencer, C. Ozturan, and G. Dundar, FISH: Fast instruction synthesis for custom processors, *IEEE Transactions on Very Large Scale Integratiions (VLSI) Systems*, vol. 20, no. 1, pp. 52–65, 2012.

[15] M. Grad and C. Plessl, Pruning the design space for just-in-time processor customization, in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, 2010, pp. 67–72.

[16] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, and K. Yelick, Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, 2008, p. 4.

[17] R. Membarth, F. Hannig, J. Teich, and H. Kostler, Towards domain-specific computing for stencil codes in HPC, in *High Performance Computing, Networking, Storage and Analysis (SCC)*, Lake City, UT, USA, 2012, pp. 1133–1138.

[18] M. Berger and J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, *Journal of Computational Physics*, vol. 53, pp. 484–512, 1984.

[19] Y. Zhang and F. Mueller, Auto-generation and auto-tuning of 3-D stencil codes on GPU clusters, in *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, 2012, pp. 155–164.

[20] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick, Implicit and explicit optimizations for stencil computations, in *ACM SIGPLAN Workshop Memory Systems Performance and Correctness*, San Jose, CA, USA, 2006, pp. 51–60.

[21] G. Rivera and C. Tseng, Tiling optimizations for 3-D scientific computations, in *Proceedings of ACM/IEEE 2000 Conference on Supercomputing*, 2000, p. 32.

[22] S. Coleman and K. McKinley, Tile size selection using cache organization and data layout, *ACM SIGPLAN Notices*, vol. 30, no. 6, pp. 279–290, 1995.

[23] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, A practical automatic polyhedral parallelizer

and locality optimizer, *ACM SIGPLAN Notices*, vol. 43, no. 6, pp. 101–113, 2008.

[24] E. Phillips and M. Fatic, Implementing the himeno benchmark with CUDA on GPU clusters, in *2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2010, pp. 1–10.

[25] Y. Yang, H. Cui, X. Feng, and J. Xue, A hybrid circular queue method for iterative stencil computations on GPUs, *Journal of Computer Science and Technology*, vol. 27, pp. 57–74, 2012.

[26] M. Araya-Polo, J. Cabezas, M. Hanzich, M. Pericas, F. Rubio, I. Gelado, and J. M. Cela, Assessing accelerator based HPC reverse time migration, *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, pp. 147–162, 2011.

[27] X. Niu, Q. Jin, W. Luk, Q. Liu, and O. Pell, Exploiting runtime reconfiguration in stencil computation, in *Proceedings of 22nd International Confererence Field Programmable Logic and Applications (FPL)*, 2012, pp. 173–180.

Yanhua Li receiced the MEng degree in computer science from Huazhong University of Science and Technology, China. He is currently a PhD candicate with the Department of Computer Science, Tsinghua University, China. His research interests include thread-level parallelism, data-intensive computing, and computer architecture.

Youhui Zhang received the BEng and PhD degrees in computer science from Tsinghua University in 1998 and 2002, respectively. He is currently a professor with the Department of Computer Science at Tsinghua University. His research interests include computer architecture, high-performance computing, and brain computing. He is a member of the IEEE and the IEEE Computer Society.

Weiming Zheng received the BEng and MEng degrees in computer science from Tsinghua University in 1970 and 1982, respectively. Now he is a professor with the Department of Computer Science at Tsinghua University, China. His research interests include high performance computing, network storage, and distributed computing. He is a member of the IEEE and the IEEE Computer Society.