Abstract

The high-performance Geospatial Information System (GIS) is expected to provide an innovative infrastructure for Earth sciences, enabling near-real-time data and processing services. Stereo-matching software has often been used in generating a Digital Elevation Model (DEM) from a pair of satellite imagery data sets to compute height from a parallax views using two photographic images. There is a need to reduce the computation time required for processing large images. We optimize stereo-matching software on multi-core/many-core processors, including Xeon, Cell and GPGPU. We describe optimization approaches of the correlation calculation part which occupied about 55% of the overall computation time. After porting and optimizing software for multi-core/many-core processors, we achieved processing time of 4.79 second (Xeon), 2.28 second (Cell) and 0.97 second (GPGPU) on each platform.

Keywords: Stereo matching, Multi-core, Many-core, Optimize, GPGPU

1. Introduction

GEO Grid [1] is a system and a concept that enables users to handle integrated processing and analysis of geographical information system data as well as various other observation data located in distributed environments, by the large-scale archiving and high-speed processing of Earth observation satellite data using grid technology. Its objective is to provide an environment in which research communities and companies can safely and securely use diverse data and make the required computations for Earth observations. One of the most important and interesting research issues is to implement the high-performance geospatial information system (GIS), which is expected to provide an innovative infrastructure for Earth sciences, enabling near-real-time data and processing services. For example, a Digital Elevation Model (DEM), which is created from a pair of satellite imagery data sets, is often used by Earth scientists to understand changes in topology. However, it normally takes from 10-15 minutes to as long as 4-6 hours to create a DEM of 4000×4000 pixels or 19000×19000 pixels, respectively, on a commodity PC. Reducing the computation time enables implementation of a near-real-time IT infrastructure based on Service Oriented Architecture (SOA). Optimization of signal processing for radio astronomy on multi-core/many-core processors has been
Table 1: Platform used in the experiments

|                        | HP Z800 Workstation          | IBM BladeCenter QS22               |
|------------------------|------------------------------|----------------------------------|
| Processor              | Xeon X5550                    | PowerXCell 8i                     |
| Clock rate             | 2.66GHz                       | 3.2GHz                           |
| #PU                    | 2                             | 2                                |
| #Core                  | 8                             | 2 PPE + 16 SPE                   |
| Memory capacity        | 4GB                           | 8GB                              |
| Memory spec            | DDR3-1333                     | DDR2-800                         |
| OS                     | Linux (Fedora 10)             | Linux (YDEL 6.1)                 |
| Compiler               | gcc 4.3.2                     | spu-gcc 4.1.1, ppu-gcc 4.1.1     |
| NVIDIA Tesla C1060     | NVIDIA Tesla C2050            |
| Processor              | Tesla                         | Fermi                            |
| Clock rate             | 1.296Ghz                      | 1.15Ghz                          |
| #PU                    | 1                             | 1                                |
| #Core                  | 240 (30 SM × 8 SP)            | 448 (14 SM × 32 CUDA cores)      |
| Memory capacity        | 4GB                           | 3GB                              |
| Memory spec            | 512bit GDDR3                  | 384bit GDDR5                     |
| Compiler               | CUDA release 2.3              | CUDA release 2.3                 |

presented[2]. In this paper, we optimize stereo-matching software, which has often been used in generating a DEM on multi-core/many-core processors, including Intel Xeon X5500 (Nehalem), IBM PowerXCell 8i (Cell), NVIDIA Tesla C1060 and C2050.

2. Evaluated platforms

Table 1 shows the platforms used in the experiments. Details of each platform are described as follows.

2.1. Intel Xeon 5500 series (Nehalem)

The Xeon 5500 series made by the Intel Corporation has an Intel x86 architecture CPU comprising a Nehalem core. It is a homogeneous multi-core processor that integrates four cores having the same performance. Because the same program operates in all cores for parallel processing, development is comparatively easy. SSE4.2 has instructions that are advantageous for the census transform method described in Section 4.2. The HP Z800 Workstation (hereafter “Z800”) is used for evaluation in this paper.

2.2. IBM PowerXCell 8i

The PowerXCell 8i made by the IBM corporation is a processor with Cell architecture, and it is suitable for media processing. Its double precision floating-point arithmetic performance is five times faster than the Cell Broadband Engine (Cell/B.E.). Because double-precision arithmetic is necessary in a DEM generator program, it is more suitable than Cell/B.E. Moreover, because Cell/B.E uses a special memory (XDR DRAM), it has a limitation in terms of memory capacity. As the PowerXCell 8i uses general-purpose DDR2 DRAM, its memory capacity can be enlarged, which makes it suitable for large-scale computation.

The PowerXCell 8i is a heterogeneous processor. It comprises one PPE core that executes general purpose instructions and eight SPE cores that execute SIMD instructions at high speed. Because the PPE is not that fast, the processing is distributed to SPE to speed it up. DMA transfer to 256KB of local storage is required for parallel programming in SPE. Optimizing on this processor requires a lot of effort compared with a homogeneous multi-core processor such as Xeon. In this paper, IBM BladeCenter QS22 (hereafter “QS22”) equipped with two CPUs is used for the evaluation.
2.3. NVIDIA Tesla C1060 and C2050

Recently, a GPGPU that uses a GPU for a general-purpose programming is attracting attention because it obtains good performance. A NVIDIA Tesla C1060 and C2050 (hereafter “C1060” and “C2050”, respectively) are installed in the Z800, and it used for the evaluation in this paper. The C1060 comprises 30 SMs (streaming multiprocessors) equipped with eight SPs (streaming processors) that operate the scalar, and has a total of 240 operation cores. The C2050 comprises of 14 SMs equipped with 32 CUDA cores that operate the scalar, and has a total of 448 operation cores.

CUDA[8] is used for programming. It divides a process into a large number of threads. It processes an operate in a unit called a grid, which collects blocks, and blocks that collects threads. It can execute two or more blocks at the same time if conditions are suitable. The number of threads that can be executed in one Block is decided by the resource that each thread uses. Moreover, it is necessary to use shared memory in each SM to increase processing speed.

3. Stereo matching

Stereo matching is processing that searches for the same points in two images using different view points, and restructures depth information using the principle of triangulation. In the case of satellite image processing, it is processing that searches for the same points in under and rear sensor images, and converts them into altitude information.

The method used to speed up the stereo matching program in this paper consists of the preprocessing part, the main processing part and the postprocessing part. The preprocessing part consists of data input, memory allocation and initializing. The postprocessing part consists of data output. Stereo matching in the main processing part consists a correlation calculation, a median filter, and interpolation of the unexpected value.

Fig. 1 shows the outline of the loop structure in the main processing part. The first loop employs coarse-to-fine method. It roughly searches for a correspondence point by using the coarse image, which is a low resolution image. The parameter of the range is changed using the result of the previous loop. It gradually searches for the correspondence point for the switch to a fine image, which is a high resolution image. The substage loop searches again for low correlation points. There is a loop-carried dependence in the coarse-to-fine method loop and the substage loop. There is no loop-carried dependence in the y axis loop or the x axis loop, and these can operated in parallel. However, when a speed increase by the recursive calculation described in Section 4.1 is implemented, the x axis loop has loop-carried dependence.

The image data used to evaluate the optimization of the stereo matching program is shown in Fig. 2. The size of these images is $3962 \times 4200$ pixels. These images are provided by the ASTER satellite sensor[3]. They are images around the Mauna Kea on the Island of Hawaii. Because a range from the sea level is covered from 0m to 4,205m in the same images, the amount of the parallax is very suitable as benchmark data.

We measured the processing time using the Z800 platform and the program before it was optimized. The correlation calculation occupied about 55% of the overall processing time. Then, we assumed that we had emphatically optimized the correlation calculation part.
4. Correlation methods

The search processing for the same point in stereo matching using the correlated calculation is described here. ZNCC is often generally used for the stereo matching of satellite images as a correlation calculation method. However, there is a problem with a high computational load. Methods such as using a low-load formula with preprocessing [4], and the recursive calculation of NCC [5] are reported to resolve this problem. In this paper, we describes a method of speeding up ZNCC using recursive calculation.

Another correlation calculation method is the census transform method [6] used for the roof detection of buildings [7]. Because an increase in speed of this method using SIMD etc. is obtained, the outline is described here.

One image is expressed as $I$ and another image is expressed as $J$. When the point in the image is expressed by vector $q$, $I(q)$ express the brightness of image $I$ in point $q$. The template is the rectangle region of image $I$. The program searches parallax vector $p$ by measuring the correlation between the template and image $J$.

$X, Y$ is the rectangular size of the template. $r = (x_r, y_r)$ is coordinate on the top left of the template. Set $W$ comprises coordinates in the template as follows:

$$W = \{(x, y)| x = x_r, \ldots, x_r + X - 1, y = y_r, \ldots, y_r + Y - 1\}. \quad (1)$$

4.1. Recursive calculation in ZNCC

ZNCC $C_{ZNCC}(p, W)$ is as follows:

$$C_{ZNCC}(p, W) = \frac{\sum_{q \in W} \left| I(q) - \bar{I}(W) \right| \left| J(p + q) - \bar{J}(p, W) \right|}{\sqrt{\sum_{q \in W} \left| I(q) - \bar{I}(W) \right|^2 \sum_{q \in W} \left| J(p + q) - \bar{J}(p, W) \right|^2}}. \quad (2)$$
The recursive calculation in ZNCC is computable as well as the case of NCC. Eq. (2) represents the calculation. Elements $M, D_I, D_J$ are calculated from $N, Q_I, Q_J, \bar{I}, \bar{J}$ which can use recursive calculation.
Figure 4: Effectiveness of parallelization by using OpenMP

Table 2: Computation time on Xeon using eight threads

| Correlation | Optimization method | Processing time (sec) | Speed increase rate |
|-------------|---------------------|-----------------------|---------------------|
| ZNCC        | None                | 40.1                  | 1                   |
|             | Recursive calculation| 4.79                  | 8.37                |
| census      | None                | 18.4                  | 1                   |
|             | SIMD instructions   | 5.4                   | 3.41                |

4.2. Census transform

The census transform method [6] uses the bit string of 0 and 1 that shows the bigness and smallness of the brightness value with a pixel at the center, and uses the Hamming distance. It is known to be robustness to brightness change and occlusion. Details of the method are as follows:

\[
\xi(p, p'; K) = \begin{cases} 
1 & K(p') < K(p), \\
0 & \text{otherwise}.
\end{cases}
\]

\[C_{\text{census}}(p, W) = \text{Hamming}\left\{\xi(q, s; I), \xi(p + q, s + q; J)\right\}.\]

Hamming\(\{a, b\}\) means Hamming distance between \(a\) and \(b\). \(s\) is a vector of center coordinates on the template; \(s = p + (X/2, Y/2)\). It is said that census transform is faster than ZNCC which requires a product-sum operation, because it can be implemented only by big and small comparing and operating the bit string. However, because a value of center coordinates is used, a recursive calculation like ZNCC cannot be used.

5. Approaches of optimization and its effectiveness

5.1. Optimization on Xeon

In optimization on Xeon, the entire operation is parallelized using OpenMP first. Afterward, we optimize it by algorithm improvement or SIMD instructions. Fig. 4 shows the effectiveness of parallelization using OpenMP. The horizontal axis shows the number of threads and the vertical axis shows speed increase rates: (speed increase rate) = (processing time in one thread) / (processing time in each thread). It is increased around 6.5 times by using eight cores.

Next, we optimize the correlation methods using the recursive calculation of ZNCC, and a census transform using SIMD instructions. The processing time is shown in Table 2. ZNCC was greatly sped up using recursive calculation. A census transform is faster than ZNCC before optimization. We optimize a census transform using SIMD instructions. The POPCNT instruction that counts the bit is added from the Nehalem core of SSE4.2. It is efficient in computing a census transform.
Table 3: Optimization process on Cell and its effectiveness

| Optimization method                        | Processing time (sec) | Speed increase rate |
|--------------------------------------------|-----------------------|---------------------|
| Original (Using only PPE)                  | 1721.29               | 1                   |
| Parallelization using SPEs                 | 153.98                | 11.18               |
| Parallelization using SIMD instructions    | 135.87                | 12.67               |
| Optimization of DMA transfer              | 34.49                 | 49.91               |
| ZNCC recursive calculation                | 30.18                 | 57.03               |
| Final (incl. optimization of pre-proc.)    | 2.25                  | 765.02              |

Figure 5: Partition of image on GPGPU

5.2. Optimization on CELL

The optimization methods in Cell (IBM PowerXCell 8i) and their effectiveness are described as follows. In Cell, the processing speed of PPE which is the core for general processing is not fast. In particular, the random access rate is very slow. However, the speed improves dramatically by the distribution of processing to SPEs and by using local storage that enables very fast access. However, it is necessary to perform optimization individually, and a lot of work is required compared with a Xeon. We implement optimization in the order shown in Table 3. The table shows the processing time and the speed-up rate in the process of each optimization phase. First, the performance improves greatly by paralleling the calculation part of ZNCC from PPE to SPEs. No too much of an improvement in performance is seen though processing SIMD instructions. In this optimization process, the DMA transfer time is a problem. The method of dividing the image in a block of constant width is used for latency hiding of a DMA transfer. Next, recursive calculation of ZNCC is implemented. The implementation of all these optimizations increases speed by a factor of about 765.

5.3. Optimization on GPGPU

We port the stereo matching program to the GPGPU using CUDA. The technique and the effectiveness of optimization are described as follows. The CUDA programing divides a process into many blocks. The image is divided as shown in Fig. 5, and we allocate one divided image in one block. The width of the partition is 32 pixels that is an efficiency value of the GPU programing. Height \( h \) decides the best value through measurement of actual performance.

Optimization includes loop-unrolling and using shared memory. The recursive calculation of ZNCC is implemented. The effectiveness of optimization on GPGPU is shown in Table 4. The processing time includes about 0.75 seconds for data transfer between the main memory on the host PC (Z800) and the global memory on the GPGPU board. The increase in speed obtained by the optimization is 2.8 to 4.28 times. In ZNCC, the recursive calculation increases speed by 6.12 times. The effect of the recursive calculation is less than the case on Cell and XEON, because the number of threads is decreased from 384 to 256.

6. Overall study

The final optimization results are shown in Table 5. With respect to the increased processing rate using the same processor, Cell is the highest because the original program porting on PPE was very slow. C2050 had the
Table 4: Optimization results on GPGPU

| Correlation | Optimization method | Width | # threads | Processing time (sec) | Speed increase rate |
|-------------|---------------------|-------|-----------|-----------------------|---------------------|
| ZNCC        | None                | 4     | 384       | 13.96                 | 1                   |
|             | Shared memory       | 6     | 384       | 3.26                  | 4.28                |
|             | Recursive calculation| 4     | 256       | 2.28                  | 6.12                |
| census      | None                | 4     | 384       | 6.86                  | 1                   |
|             | Shared memory       | 12    | 384       | 2.45                  | 2.80                |

Table 5: Overall performance

| Correlation | Platform         | Original (sec) | Optimized (sec) | Speed increase rate |
|-------------|------------------|----------------|-----------------|---------------------|
| ZNCC        | Xeon (Z800)      | 236.38         | 4.79            | 37.51               |
|             | Cell (QS22)      | 1721.29        | 2.25            | 765.02              |
|             | GPGPU (C1060)    | 13.96          | 2.28            | 6.12                |
|             | GPGPU (C2050)    | 1.47           | 0.97            | 1.52                |
| census      | Xeon (Z800)      | 152.12         | 5.37            | 28.17               |
|             | Cell (QS22)      | 1031.30        | 2.49            | 414.18              |
|             | GPGPU (C1060)    | 6.86           | 2.45            | 2.8                 |
|             | GPGPU (C2050)    | 2.27           | 0.83            | 2.74                |

fastest processing, but the speed increase rate was lowest because the original program was already parallelized using CUDA. The original C2050 program is very fast because C2050 includes a cache memory. Although, optimized program using shared memory is faster than original program. QS22 equipped with two processors showed almost equal performance to C1060 equipped with one processor. Z800 had about half the performance of QS22.

A summary of the optimization approaches and the results is shown in Table 6. The findings show optimization in CELL requires of the most work, while because OpenMP can be used for automatic parallelization, the work required for Xeon is the least. The work required for GPGPU lies in the middle of these two.

7. Conclusion

We ported and optimized a stereo matching program in DEM generation using many-core processor platforms, and evaluated the differences and effectiveness of the optimization techniques on each platform. Two correlation methods, ZNCC and census transform, were evaluated in this paper. Moreover, a high-speed recursive calculation method was implemented in ZNCC. In optimization on Xeon, we used OpenMP, a recursive calculation on ZNCC, and SIMD instructions on census transform. In the optimization on Cell, we ported from a low-speed PPE to SPE, and optimizing the DMA transfer.

In optimization on GPGPU, the optimum size of block was properly requested. The performance of C2050 was 2.35 times (ZNCC) and 2.95 times (census) faster than C1060. The performance of two Cells was almost the same as C1060. The performance of two Xeons was half of two Cells.

Our future work will include clustering that assumes the use of high-resolution satellite images.

References

[1] S. Sekiguchi, Y. Tanaka, I. Kojima, N. Yamamoto, S. Yokoyama, Y. Tanimura, R. Nakamura, K. Iwao and S. Tsuchida: Design Principles and IT Overview of the GEO Grid, IEEE Systems Journal, Vol.2, No.3, pp.374-389 (2008).
[2] R. V. Nieuwoort and J. W. Romein: Using Many-Core Hardware to Correlate Radio Astronomy Signals, Proc. ICS’09, pp. 440-449, June 8-12 (2009).
[3] Y. Yamaguchi, B. A. Kahle, M. Pniel, H. Tsu, and T. Kawakami: Overview of Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER), IEEE Trans. Geosci. Remote Sens., vol. 36, no. 4, pp. 1062-1071 (1998).
[4] L. D. Stefano, M. Marchionni and S. Matteo: A fast area-based stereo matching algorithm, Image and Vision Computing, Volume 22, Number 12, pp.983-1005 (2004).
Table 6: Summary of the optimization approaches and the results

|                  | Xeon (Z800) | Cell (QS22) | GPGPU (C1060 & C2050) |
|------------------|-------------|-------------|-----------------------|
| Pallalization    | OpenMP      | Manual porting | CUDA                 |
| SIMD instruction | SSE         | SIMD on SPEs | None                  |
| Optimization of memory access | None   | Latency hiding of DMA | Using shared Memory |
| Term of work     | 1 week      | 4 weeks     | 2 weeks               |
| Amount of optimized source code | 500 lines | 920 lines   | 550 lines             |

[5] O. Faugeras et al. : Real time correlation-based stereo: Algorithm, implementations and applications, INRIA Technical Report, RR-2013 (1993).
[6] R. Zabih and J. Woodfill : Non-parametric Local Transforms for Computing Visual Correspondence, Proc. ECCV’94, pp.151-158 (1994).
[7] D. Woo and D. Park : Rooftop Detection Based on 3D Line Data Using Fast Graph Search, Ninth International Conference on Hybrid Intelligent Systems, pp.442-446 (2009).
[8] M. Harris : Optimizing Parallel Reduction in CUDA, http://developer.nvidia.com/ (2007).