Compilation Time-Based Analysis using Optimized Iterative Techniques

Ume Farwa[1], Khurshid Asghar[2†], Mubbashar Saddique[2]
umefarwa37@gmail.com, khasghar@uo.edu.pk , mubashar.chaudary@uo.edu.pk
[1]Department of Computer Science, Information Technology University Lahore 54000, Pakistan
[2]Department of Computer Science, University of Okara 56300, Pakistan

Abstract
Compilation time has always been an important factor for performance analysis of any system. This paper discusses the various optimized iterative techniques to analyze the performance of programs like loop unrolling, loop level parallelism or loop-carried dependence, and loop ordering. In first technique, loop is unrolled up to scale 5 and then compared with rolled one to find out performance differences. In the second technique, it finds that the loop carried dependence and inter-change the statements order and exposes the par-allelism. In the third one, the loop order is changed to reduce the jump calls during code execution. The execution time of all three methods are compared, that is the proof of high performance after implementing the optimized iterative techniques. The execution time may differ on different machines. The results are calculated on a core i5 machine with 2.7Ghz processor under Linux kernel.

Key words:
Compilation time; performance analysis; iterative techniques; program optimization.

1. Introduction

Optimization techniques are the helping hand of any system developed to perform well. Different optimization techniques are developed to reduce the execution time/ memory usage by processor. Some of these are implemented on ma-chine level, while others are the source level implementation. This paper discusses the source level optimization of iterative methods to reduce the execution time of a program. The first technique discusses the loop unrolling against loop rolling. If a large number of loop iterations exist in the kernel pipeline, the loop iterations could potentially be the critical path of the kernel pipeline. UL can increase the pipeline throughput by allocating more hardware resources to the loop [1]. It causes the reduction of compilation time that indirectly adds up to performance. The loop-carried dependence is the iterative dependency that exists in loops that causes a barrier to implement parallelism. To implement the loop-level parallel-ism, we need to recognize the structure of loops, arrays and any variable involved. If the findings show that the statements inside the loop are not circular dependent, then we can make alterations in statements to execute parallel and improve the execution time. The results of this technique also count for higher performance and reduction in compilation technique [2]. The third technique involves changing the loop order. The impact of loop order is an important count in execution time. Basically, it reduces the jump calls between instruction execution which in return reduce the overall execution time. It comes up with vertical and horizontal execution order of instructions.

$$
\begin{array}{l}
for\ (int\ k\ =\ 0;\ k\ <\ 100;\ k++) \\
\quad \{ \\
\quad \ldots \\
\quad for\ (int\ j\ =\ 0;\ j\ <\ 100;\ j++) \\
\quad \quad \{ \\
\quad \quad \ldots \\
\quad \quad \} \\
\end{array}
$$

$$\Downarrow$$

$$
\begin{array}{l}
\quad for\ (int\ j\ =\ 0;\ j\ <\ 100;\ j++) \\
\quad \quad \{ \\
\ldots \\
\quad for\ (int\ k\ =\ 0;\ k\ <\ 100;\ k++) \\
\quad \quad \{ \\
\ldots \\
\quad \quad \} \\
\quad \quad \}
\end{array}
$$

This technique also counts for the high-performance system that needs the iterative solutions to be implemented. The paper is further divided as follows. It discusses related work that is already gone through the experiment. Then the methodology is explained under all the techniques experimented during this research. Then the results and conclusion sum up the discussion about the experiment under discussion.

2. Related Work

Loop unrolling is widely helpful in different types of applications. Some of its applications exist in image processing where the image convolution takes help of loop unrolling while multiplying the matrix. It helps in creating optimized algorithms. The performance after optimization

Manuscript received May 5, 2020
Manuscript revised May 20, 2020

and parallelism speeds up over 2000x over baseline [3]. The work has been done on parallel frameworks that offer the programming patterns which express the concurrency in applications that enables the usage of hardware in parallel manners. This transforms the sequential instructions to parallel after identifying the map and pipeline the parallel patterns. [5] The kernel level optimization is also done by finding the loop pattern for its different activities. The loop order is an important factor that always impacts the performance as well as compilation time [4].

3. Proposed Methodology

This approach comes up with three different methodologies to find out the best possible optimization on source level. All the three techniques are explained here with source code and results. Every function provokes from the $main()$ function. While for comparing the results both scenarios are discussed.

3.1 Loop unrolling

The following given code is about loop rolling and unrolling.

$$Loop\ rolled:$$

$$void\ loopRolled\ (\ )$$
$$\{$$
$$int\ x\ =\ 0, y\ =\ 0;$$
$$for\ (int\ i\ =\ 0;\ i\ <\ 100000000;\ i++)$$
$$\{$$
$$x\ =\ y;$$
$$\}$$

$$Loop\ unrolled:$$

$$void\ loopUnrolled()\ \{$$
$$int\ x\ =\ 0, y\ =\ 0;$$
$$for\ (int\ i\ =\ 0;\ i\ <\ 20000000;\ i++)$$
$$\{$$
$$x\ =\ y;$$
$$x\ =\ y;$$
$$x\ =\ y;$$
$$x\ =\ y;$$
$$x\ =\ y;$$
$$\}$$
$$\}$$

Loop is unrolled up to scale 5. What we did is just replace the repetitive instruction with the same 5 instructions and lower the number of iterations five times of total.

3.2 Loop level parallelism

The following code is about eliminating the loop-carried dependence and exposing the loop-level parallelism [2].

$$Loop-level\ dependence:$$

$$void\ LLP1()$$
$$\{$$
$$int\ A[100], B[100+1], C[100], D[100];$$
$$for\ (int\ j\ =\ 0;\ j\ <\ 1000000;\ j++)$$
$$\{$$
$$for\ (int\ i\ =\ 0;\ i\ <\ 100;\ i++)$$
$$\{$$
$$A[i]\ =\ A[i]\ +\ B[i];$$
$$B[i\ +\ 1]\ =\ C[i]\ +\ D[i];$$
$$\}$$
$$\}$$
$$\}$$

$$Loop-level\ parallelism$$
$$void\ LLP2()$$
$$\{$$
$$int\ A[100], B[100+1], C[100], D[100];$$
$$for\ (int\ j\ =\ 0;\ j\ <\ 1000000;\ j++)$$
$$\{$$

```
A[0] = A[0] + B[0];
for (int i = 0; i < 99; i++)
{
B[i + 1] = C[i] + D[i];
A[i + 1] = A[i + 1] + B[i + 1];
}
}
B[100] = C[99] + D[99];
}
```

The above code is just finding out the dependency between the instructions and finding if it can be removed or not.

3.3 Loop ordering

The following code describes the loop ordering technique that is also helpful in some scenarios to enhance the performance.

```
Loop order 1:
void loop1()
{
int x[100][100];
for (int i = 0; i < 10000; i++)
{
for (int j = 0; j < 100; j++)
{
for (int k = 0; k < 100; k++)
{
x[j][k] = i;
}
}
}
```

```
Loop order 2:
void loop2()
{
int x[100][100];
for (int i = 0; i < 10000; i++)
{
for (int k = 0; k < 100; k++)
{
for (int j = 0; j < 100; j++)
{
x[j][k] = i;
}
}
}
}
```

It can be observed that here the most inner loop is interchanged with the second inner loop thus improves the execution time.

3.4 Execution process

The execution is the process where we can compare the results of its compilation time that indirectly is a sign of performance evaluation of the system. To get the execution time, we come up with the following strategies.

```
// Get the system time before function start
auto start = high_resolution_clock::now();
// Invoke the function,
loop1()
//Get the system time after function stop execution
autostop = high_resolution_clock::now();
// Get the total time duration.
```

$auto\ duration\ =\ duration\_cast\ <microseconds>\ (stop - start)$;

4. Results

The results are taken from two different sources after compilation. The first source is the online compiler [6]. The second source is the Linux OS. Table1 shows the results.

LR: Loop rolling

LUR: Loop unrolling

LLP: Loop-level parallelism

Table 1: Margin specifications

LR()	LUR()	LLP1()	LLP2()	Loop1()	Loop2()
1.05	0.3001	4.3828	1.8878	1.4375	0.3556
0.2126	0.0421	3.3843	2.903	2.5339	2.3422

5. Conclusion

The techniques for optimizing the iterative methods are useful scenario to scenario. For example, some techniques might not be as helpful as expected. On the other hand, there is machine to machine performance variation. So, the above results may vary if the same code is run on some other machine with different environments. Overall these techniques are helpful in various real time applications as well as for the OS itself, as stated above. Image processing takes it into a great account to use the loops for matrix manipulation. These techniques are still on the way to improve time to time for achieving high performance on different systems.

References

[1] Z. Wang, B. He, W. Zhang, and S. Jiang, "A performance analysis framework for optimizing OpenCL applications on FPGAs," in 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016, pp. 114-125.

[2] J. L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach vol. 6: Elsevier, 2019.

[3] A. Tousimojarad, W. Vanderbauwhede, and W. P. Cockshott, "2D Image Convolution using Three Parallel Programming Models on the Xeon Phi," arXiv preprint arXiv:1711.09791, 2017.

[4] T. M. Low, F. D. Igual, T. M. Smith, and E. S. Quintana-Orti, "Analytical modeling is enough for high-performance BLIS," ACM Transactions on Mathematical Software (TOMS), vol. 43, pp. 1-18, 2016.

[5] D. del Rio Astorga, M. F. Dolz, L. M. Sánchez, J. D. García, M. Danelutto, and M. Torquati, "Finding parallel patterns through static analysis in C++ applications," The International Journal of High Performance Computing Applications, vol. 32, pp. 779-788, 2018.

[6] (2020, 15-Jan-2020). C++ Debugger. Available: https://www.onlinegdb.com/

Ume Farwa completed BS (Information Technology) from University of Education, Lahore in 2017. Presently, she is MPhil Scholar at Information Technology University Lahore, Pakistan. Her research interest is including HCI, machine learning, data mining, networks and programming.

Khurshid Asghar is working as Associate Professor at Department of Computer Science University of Okara. He earned PhD degree in the field of image forensics from COMSATS University Islamabad, Pakistan. He also worked as research associate at Cardiff School of Computer Science and Informatics, Cardiff University, UK. His current research interest includes Image Processing, Image and Video Forensics, Machine Learning, Deep Learning, Network Security, Biometrics, Medical Imaging Brain Signals, Geometric Modeling and Computer programming.

Mubbashar Siddique is working as Lecturer at Department of Computer Sciences. He completed BSc (Telecommunication Engineering) from Institute of Engineering & Technology, Lahore Campus, and Pakistan. He got merit scholarship from COMSATS University Islamabad (Abbottabad Campus), Pakistan where he completed his MS computer science in 2010. Presently, he is a PhD Scholar at COMSATS University Islamabad, Pakistan. Mr. Siddique also worked as a research associate at Department of Cyber Defense Graduate School of Information Security, Korea University, South Korea. Presently, he is working in video and image forensic domain. Furthermore, his research interest is in the area of image/video processing, computer vision, machine learning, data mining and networks.