Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

30.11.2012
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

Outline

1. Introduction
2. Approach
3. Details
4. Conclusions
Evolution of Multi-core Processors

In the context of desktop computers

(a) 2000: 1 core
(b) 2005: 2 cores
(c) 2006: 4 cores
(d) 2011: 8 cores

⇒ Parallel programming is no longer optional for increased speed-up

Challenges of Automating Parallelization

- Synchronization $\Rightarrow$ Deadlocks, Starvation, etc.
- Detection of promising parallelization regions
- Non-faulty transformation

0 Acknowledgements to FreeDigitalPhotos.net and Sira Anamwong
Requirements

- Automatic or assisting tools
- Program restructuring
- Dependence analysis
Related Work

- Loops and arrays \([\text{ROR}^+08, \text{YTT}^+00]\), rarely I/O access
- Static \([\text{HAM}^+05]\) or dynamic analysis, rarely both
- C or Java byte code \([\text{FGN12}]\), rarely Java source code
- No business applications for evaluation

⇒ Goal: A more high-level approach
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Figure 1: Overview of the approach
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Legend:
- S1: SDG construction
- S2: Gathering
- S3: SDG Enrichment
- S4: Ranking
- S5: Pattern detection
- S6: Transformation

Figure 2: S1: SDG construction
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Figure 3: S2: Gathering runtime information
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Figure 4: S3: SDG enrichment
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Legend:
S1: SDG construction
S2: Gathering
S3: SDG Enrichment
S4: Ranking
S5: Pattern detection
S6: Transformation

Figure 5: S4: Ranking
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Figure 6: S5: Pattern detection
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Figure 7: S6: Semi-automatic transformation
Figure 8: More high-level: Parallelize I/O accesses automatically
Distinguish I/O accesses

An Example

```java
for (int i = 0; i < a.length; i++) {
    FileInputStream fis = new FileInputStream(a[i]);
    int fileContent = fis.read();
    ...
}
```

constructor of I/O type

filename as argument
Distinguish I/O accesses

An Example

```java
for (int i = 0; i < a.length; i++) {
    FileInputStream fis = new FileInputStream(a[i]);
    int fileContent = fis.read();
    ...
}
```

file operation
Detection and Utilization of Potential Parallelism in Software Systems

Christian Wulf

1. Introduction
2. Approach
3. Details
4. Conclusions

Figure 9: Exemplary candidate pattern
1. Introduction
2. Approach
3. Details
4. Conclusions

Figure 10: Exemplary candidate pattern
Figure 11: One associated parallelization pattern
Conclusions

Figure 12: Software lags behind hardware development

- No standard supporting parallelization tools available
- Pattern-matching approach to assist in parallelization

Acknowledgements to Freddigitalphotos.net and digitalart
References

www.freedigitalphotos.net.

Min Feng, Rajiv Gupta, and Iulian Neamtiu.
Effective parallelization of loops in the presence of i/o operations.

Mary W. Hall, Saman P. Amarasinghe, Brian R. Murphy, Shih-Wei Liao, and Monica S. Lam.
Interprocedural parallelization analysis in suif.

Easwaran Raman, Guilherme Ottoni, Arun Raman, Matthew J. Bridges, and David I. August.
Parallel-stage decoupled software pipelining.

Chao-Tung Yang, Shian-Shyong Tseng, Chang-Jiun Tsai, Cheng-Der Chuang, and Sun-Wen Chuang.
A new model of exploiting loop parallelization using knowledge-based techniques.