Advanced Compilation Techniques for Data Distribution and Chip Multiprocessor

Amruta Salunkhe, Prof, R. D. Bharati, Dr.D.Y.Patil Institute Of Engg And Technology,Pune

Abstract— The limiting factor in the performance of chip multiprocessors is data access latency. Data access latency increases significantly as the number of cores in non uniform architecture are increased. To moderate this effect, we use a compiler-based approach to leverage data access locality. Multithreaded memory access patterns (MMAPs) are resolved and used by a partitioning algorithm to choose a partition of allocated memory blocks among the forked threads in the analyzed application. This partition is used to impose data ownership by associating the data with the core that executes the thread owning the data. Based on the partition, the communication pattern of the application can be extracted.

We show how this data can be used in an experimental architecture to speedup applications. Compiler assisted data partitioning method shows a 20 percent speedup over shared caching and 5 percent speedup over the closest runtime approximation. By using a communication pattern, a comparable performance is achieved compared to a system that uses only compiler generated partition and uses a complex centralized network configuration system at runtime.

Index Terms— Data Partitioning, Multi-Threaded Memory Access Pattern, TI Variable, Access Latency, Chip Multiprocessor, Multithreaded.

1 Introduction

Multithreaded benchmarks shows that a large portion of data parallel applications tend to exhibit regularity in their data access patterns. Cache architectures for the last level on-chip cache for tiled CMPs typically fall into two classes, non-uniform cache access (NUCA) and private access with the higher-levels typically being private. Many multithreaded applications, such as those from the SPLASH-2 and PARSEC benchmark suites, exhibit regular data access patterns that can be exploited at compile time.

Detecting these patterns is critical to gain an insight of how data should be distributed among multiple CMP tiles to promote the locality of access within a tile and between tiles implied in the program. MMAPs are primarily related to array accesses and TI-Structures, there are other programming components affecting the actual access
patterns. Several runtime oriented caching policies
and schemes have been proposed to distinguish certain characteristics of data access patterns in order to achieve better performance.

The existing system have Compiler analysis for determining application memory access and communication behavior in single and multiprocessor systems. Cache design to promote locality of access and low-latency access. Interconnect design to promote low-latency and/or high throughput communication. But that system doesn’t involves catch-block granularity.

We suggest to use compiler to determine data partitioning implied by the program. Partitioning will be communicated to runtime system to help to determine data placement in cache to promote the locality of access. Based on the partitioning, communication pattern of the application will be determined.

The compilation methodology requires the use of several known compiler techniques that are well understood, such as symbolic analysis, constant propagation, expression folding, etc. We propose to use the compiler to determine the data partitioning implied by the program. Compiler analysis techniques to detect data access patterns, partitions, and communication patterns for multithreaded applications. System saves significant runtime complexity. Achieves an 5.1 percent additional speedup through the addition of the reconfigurable network.

2 Related work

Non uniform cache access designs solve the on-chip wire delay problem for future large integrated caches by embedding a network in the cache, NUCA designs let data migrate within the cache, clustering the working set nearest the processor [1].

Princeton Application Repository[2] for Shared Memory Computers (PARSEC), is a benchmark for studies of Chip-Multiprocessors(CMPs). Earlier benchmarks for multiprocessors focused on high performance computing applications and used a limited number of synchronization methods. PARSE considers upcoming applications in recognition, mining and synthesis (RMS) and systems applications which mimic large scale multithreaded commercial programs[2].

Analysis of energy models for on-chip interconnection networks and tradeoffs in tiled chip multiprocessors design. Using these models, investigation of the network architecture including topology, channel width, routing strategy, and buffer size done to analyze how it affects performance and impactarea and energy efficiency. Performance of on-chip networks designed for tiled chip multiprocessors implemented in an advanced VLSI process analyzed and compare area and energy efficiencies estimated from this models[3].

It describes an automatic parallelization system for Fortran programs and also presents the results obtained with the extensive experiments. The system incorporates a comprehensive and integrated collection of analyses including dependence, privatization and reduction recognition for both array and scalar variables, and scalar symbolic analysis to support these[4].

2.1 Data Partitioning

Consider a parallel program for four threads. The compiler detects that the program has three phases. It can be determined from the analysis that multi-threaded access patterns of each phase as follows.

Algorithm

```
Input  : File F, Number of Processor P
Output: Allocated List
  Step 1: Read the file
  Step 2: Analyze the size
  Step 3: Get speed of Processor
  Step 4: Compute the Splited part
  Step 5: while (Splited JobList)
          {
              SJ; =SJList.get (i);
              Allocated to Processor P; & Create Thread
              Remove.SJ(i) from SJList();
          }
   Step 6: Execute the thread
```

2.2 System Overview

To achieve the data partitioning using compiler assisted program, a reconfigurable system is designed. Fig 1. Shows a full system overview which is organized in a two dimensional grid where each tile contains a processor core, a private L1 instruction and data cache, a L2 cache bank, a directory bank and a network interface (NI). Communication between tiles can happen via one of the four two dimensional mesh network planes each of which combine circuit switching and packet switching. Routers of these planes are also given in fig 1.

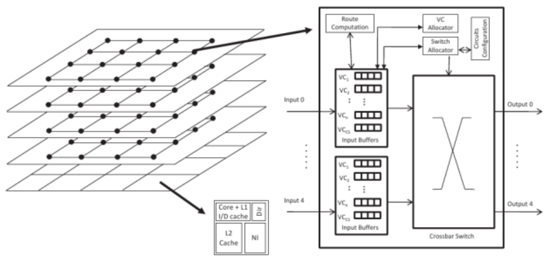

Fig.1 System Overview

3 PROGRAMMER MODELS

If array access appears N times in a m-deep nested loop, and the lower bounds, upper bounds, and steps of loop indices for thread x are ι1(p),… ιm(p), u1(p),… u m(p), and s1(p),… sm(p), respectively. Hence, the formula to calculate access weight for R(p)

$$W(R(p)) = N * \prod_{K=1}^{M} \frac{(U\alpha(p) - L\alpha(p))}{S\alpha(p)}$$

To estimate the nonlocal access weight W, first need to compute the ending offset, overlap range and destiny of a region. Consider a region Ri(x) that has m span stride pairs. Sik(p) and Tik(p) are the Kth span stride within the region and O*i*(p) are the region's initial offset. Define the corresponding ending offset Ei(p) as the last element in the region.

$$Ei(p) = Oi(p) + \sum_{K=1}^{m} Sik(p).$$

Thus, the overlap range of Ri(p) and R*(q) is defined as

$$\Delta = min(Ei(p), E^*(q)) - max(Oi(p), O^*(q)))$$

Where O*(y), E*(y) are the initial offset and ending offset respectively, of R*(y)

The region R*(q)'s density D*(q) is the access weight of region relative to the range relative to the range of that region. Thus, D*(q) is defined as

$$D^*(q) = \frac{1}{E^*(q) - O^*(q)} * W(R^*(q))$$

We thus approximate nonlocal access weight of tile p to data in R*(q) (equivalent to amount of communication) in phase i as follows

$$W(p,q,i) = \frac{\Delta}{Ei(p) - Oi(p)} * D^*(q) * W(Ri(p))$$

Number of Split NS = 4;

Number of Processor NP = 4;

Speed of Processor PS

Size of Data SD = 1000 MI (Million instruction) or 100 kb

$$SD_S = \frac{S_n}{\sum_{i=0}^{N_P} P_{Si}}$$

Split Part for Particular Processor = $SD_S * P_{Si}$

Processor	Speed
P1	10
P2	5
P3	10
P4	15

SD_S = 100/(10+5+10+15)

SD_S = 100/40 = 2.5

Processor	Stored Split Data
P1	25
P2	12.5
P3	25
P4	37.5

4 Conclusion

In this project, we have developed compiler analysis techniques to detect data access patterns, partitions, and communication patterns for multithreaded applications. The data discovered is then used to assign ownership to subpages, and this data is then used by the CAP caching scheme for data distribution. The CAP is used to compute the communication pattern which can be used as a method to program a configurable interconnect (CAC).
We evaluated the proposed method on various benchmarks from the SPLASH-2 and PARSEC benchmark suites and compared the results with other relevant schemes. The experimental results shows that the compiler-assisted caching scheme inherits the capacity benefit of distributed shared caches while reducing the memory access latency.

5 Result

Graph showing comparison between existing and proposed system. Results are showing decrease in size with the proposed system while also showing around 20% decrease in data latency as well.

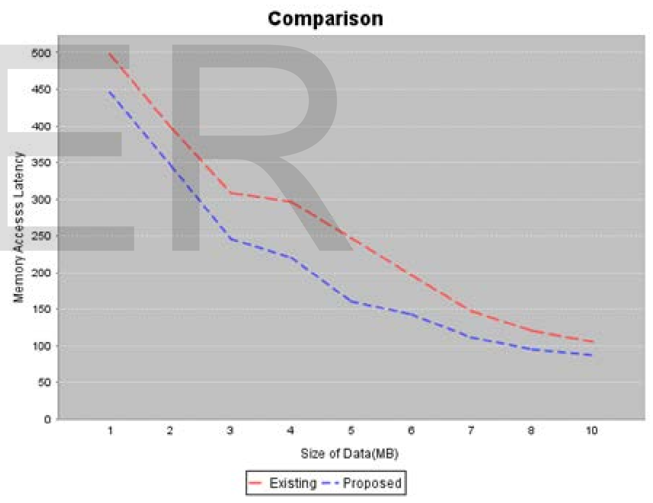

Fig 2. Comparison between existing & proposed system

Graph showing comparison between existing and proposed system in terms of speed. New system is showing increase in speed of about 20% with the new system.

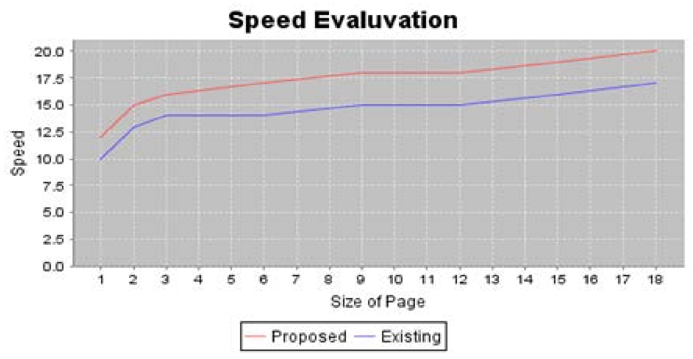

Fig 3. Speedup Evaluation

REFERENCES

[1] Yong Li, Ahmed Abousamra, Rami Melhem, and Alex K. Jones, "Compiler-Assisted Data Distribution and Network Configuration for Chip Multiprocessors" IEEE Trans. Parallel and Distributed Systems VOL, 23 NO.11,NOV 2012.

[2] C. Bienia, S. Kumar, J.P. Singh, and K. Li, “The Parsec Benchmark Suite: Characterization and Architectural Implications,” Technical Report TR-811-08, Princeton Univ., Jan. 2008

[3] J.D. Balfour and W.J. Dally, “Design Tradeoffs for Tiled cmp On-Chip Networks,” Proc. 20th Ann. Int’l Conf. Supercomputing (ICS),pp. 187-198, 2006.

[4] Z. Li and P. Yew, “Efficient Interprocedural Analysis for Program Parallelization and Restructuring,” Proc. SIGPLAN Symp. Parallel Programming: Experience with Applications, Languages and Systems, July 1988.

[5] B. Creusillet and F. Irigoin, “Exact versus Approximate Array Region Analyses,” Proc. Ninth Int’l Workshop Language and Compilers for Parallel Computing, Aug. 1996.

[6] Y. Paek, E.Z.A. Navarro, J. Hoeflinger, and D. Padua, “An Advanced Compiler Framework for Noncache-coherent Multiprocessors,” IEEE Trans. Parallel and Distributed Systems, vol. 13, no. 3, pp. 241-259, Mar. 2002.

[7] C. Kim, D. Burger, and S.W. Keckler, “Nonuniform Cache Architectures for Wire-Delay Dominated On-Chip Caches,” IEEE Micro, vol. 23, no. 6, pp. 99-107, Nov./Dec. 2003.