Memory Management for Data Localization on OSCAR Chip Multiprocessor

Hirofumi Nakano†, Takeshi Kodaka†, Keiji Kimura‡, Hironori Kasahara†
{hnakano,kodaka,kimura,kasahara}@oscar.elec.waseda.ac.jp
†Department of Computer Science,
School of Science and Engineering, Waseda University,
‡Advanced Research Institute for Science and Engineering, Waseda University,
3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555, Japan

1 Introduction

Recently, chip multiprocessor architecture that integrates multiple processor cores on a chip is getting popular like Power4 [1], UltraSPARC IV [2] and OSCAR chip multiprocessor (OSCAR CMP) [3] for multigrain parallel processing. In order to control data allocation by the multigrain parallelizing compiler, OSCAR CMP has local data memory (LDM) for processor private data and distributed shared memory(DSM) having two ports for processor shared data. With the increasing gap between processor and memory access speeds, effective use of local memory is getting more important. This paper describes how to manage LDM on OSCAR CMP for data localization which exploits data locality by assigning coarse grain tasks sharing the same data onto the same processor consecutively. This scheme is implemented on OSCAR multigrain parallelizing compiler [4]. The proposed scheme is evaluated on OSCAR CMP, using Swim and Tomcatv from the SPEC fp 95.

In this paper, Section 2 describes coarse grain task parallel processing. Section 3 proposes local memory management on OSCAR CMP for data localization. Section 4 evaluate the performance of the proposed schemes on OSCAR CMP using Swim and Tomcatv from the SPEC fp 95 benchmark suite. Concluding remarks are described in Section 5.

2 Coarse Grain Task Parallel Processing

This section describes coarse grain task parallel processing, which is a part of multigrain parallel processing.

Coarse grain task parallel processing uses parallelism among three kinds of macro-tasks(MTs), or coarse grain tasks, namely, block of pseudo assignment statements (BPA), repetition block (RB) and subroutine block (SB).

The compiler decomposes a source program into macro-tasks. Also, it generates macro-tasks hierarchically inside a sequential repetition block and a subroutine block.

Coarse grain task parallelization by OSCAR compiler is performed in the following steps.

1. Decomposition of a source program into macro-tasks.
2. Analysis of data and control flows among macro-tasks and generation of Macro Flow Graph (MFG) representing data and control flows.
3. Analysis of Earliest Executable Condition (EEC) based on data and control dependence analysis that represents the condition, on which macro-task may start its execution earliest, and generation of Macro Task Graph (MTG) that represents the EEC.
4. Scheduling macro-tasks to processors or processor groups.

When a macro-task graph has no conditional dependencies, macro-tasks are statically scheduled to processors or processor groups at a compiler time and parallelized code is generated for each processor according to the scheduling results. When macro-task graph contains control dependencies, compiler generates dynamic scheduling routine to assign macro-tasks to processors or processor clusters at a run time and embeds the dynamic scheduling routine to the generated parallelized code with macro-task code in order to cope with runtime uncertainties.

3 Memory Management for Data Localization on OSCAR CMP

In this section, at first, OSCAR CMP architecture is described. Next, the proposed memory management and data transfer calculation are explained.

3.1 OSCAR CMP

Figure 1 shows an OSCAR CMP architecture. OSCAR CMP has multiple PEs on a chip. Each PE has a

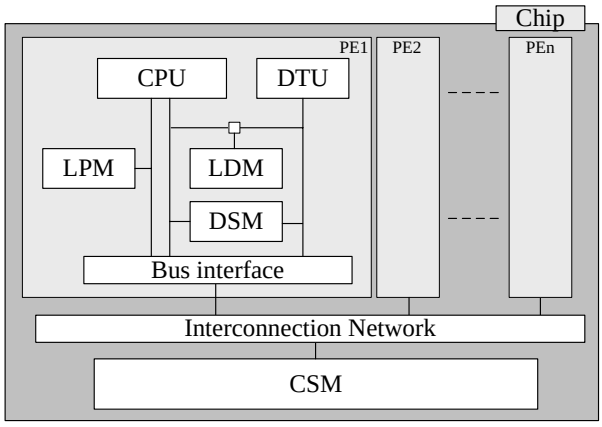

Figure 1: OSCAR Chip Multi Processor

simple single issue in-order processor core, Local Data Memory(LDM) having one port for processor private data, Distributed Shared Memory(DSM) having two ports for shared data, Local Program Memory(LPM) for program code, and Data Transfer Unit(DTU) for asynchronous data transfer. All PEs on a chip are connected by Interconnection network like bus and crossbar. In addition, Centralized Shared Memory(CSM) is integrated on the chip.

3.2 Data Localization

In order to exploit data locality from a program and use LDM on each PE efficiently, the data localization with Loop Aligned Decomposition [5] is applied. The LAD processes on multiple RBs like doall loops and reduction loops.

In the data localization, first, RBs on the critical path, their preceding RBs and succeeding RBs are selected and grouped into a Target Loop Group(TLG) to which the LAD is applied. Next, macro-tasks in the TLG are decomposed so that shared data can be passed through LDM considering LDM size. Data Localization Group(DLG) is defined as the group of macro-tasks to be assigned to the same processor consecutively to localize the decomposed arrays.

3.3 Scheduling considering Data Localization Groups

After LAD, macro-tasks are scheduled onto PEs statically in this paper. The LAD decomposes MTs so that all macro-tasks in the same DLG can be assigned to the same processor consecutively. The scheduler is developed based on ETF/CP/MISF [6], and made some modification in the scheduling priority. Macro-tasks inside of DLG are assigned to the same processor as consecutively as possible, and macro-tasks outside of DLG are assigned not to disturb it.

3.4 Local Memory Management

Figure 2 shows an image of memory map of LDM on each PE. As shown in this figure, a memory area on LDM, named DLG_Slot is allocated for each TLG. The

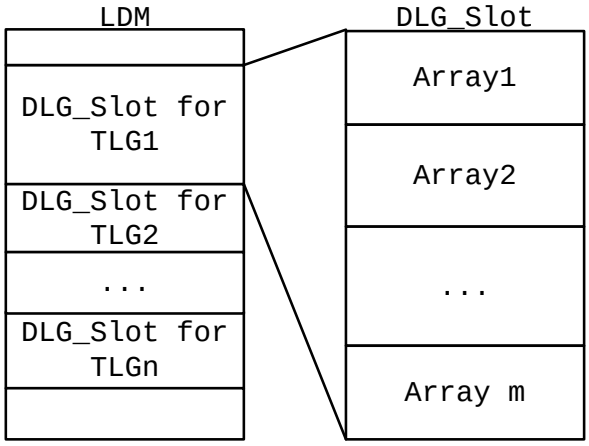

Figure 2: Local Memory Map

arrays which are decided to located on LDM, are assigned these DLG_Slot. As described in section 3.3, the static scheduler assigns all macro-tasks in the same DLG consecutively onto one PE, so that all localized arrays in the macro-tasks which belong to the same DLG can be assigned to one DLG_Slot.

3.5 Data Transfer Calculation

All localized arrays in the macro-tasks which belong to the same DLG are passed through LDM. Therefore data transfers among such macro-tasks are unnecessary. Data transfers between MT_i ($1 \leq i \leq$ total number of macro-tasks) inside of DLG_j ($1 \leq j \leq n$. Here, each macro-task in the TLG is decomposed into n sub macro-tasks) and MT_k ($1 \leq k \leq$ total number of macro-tasks) outside of DLG_j, similarly, data transfers between MT_i and $Layer_{outside}$ which is outside of target MTG are necessary. How to calculate data transfers among relationships is described here. First, array def-use chains(DU_Chain[producer][consumer]) among such relationship are calculated. Then, load and store of MT_i are represented as following.

$$load[MT_i] = \cup(DU\_Chain[l][MT_i] - DU\_Chain[l][pred])$$

Here, l means macro-tasks inside of DLG_j or $Layer_{outside}$, and $pred$ means macro-tasks inside of DLG_j and predecessor of MT_i.

$$store[MT_i] = \cup(DU\_Chain[MT_i][l])$$

Here, l means macro-tasks outside of DLG_j.

4 Performance Evaluation

In this section, the performance of the proposed scheme is evaluated on OSCAR CMP using Swim and Tomcatv from the SPEC fp 95.

4.1 Evaluation Environment

The proposed scheme is evaluated using Swim and Tomcatv from SPEC fp 95. To reduce simulation time, number of iterations in each loop is reduced. The

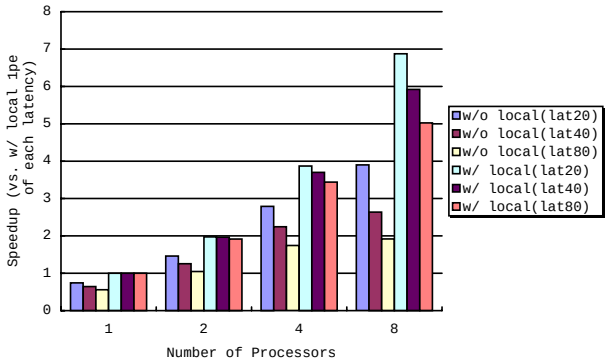

Figure 3: Speedup of Swim

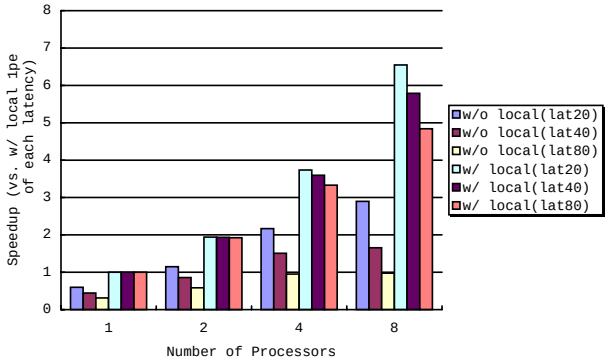

Figure 4: Speedup of Tomcatv

whole data size of Swim and Tomcatv in this evaluation are about 3.3MB and 230KB respectively. The evaluated memory latency set of OSCAR CMP are described in Table 1. The detailed architecture simulator is used for these evaluations.

Table 1: Evaluated Memory Latency of OSCAR CMP

	lat20	lat40	lat80
CSM latency	20	40	80
LDM latency	1	2	4
DSM latency	1	2	4
LPM latency	1	1	1

4.2 Performance

Performance evaluation results for Swim and Tomcatv are shown in Figure 3 and 4 respectively. These figures show speedup against sequential execution clocks by the proposed scheme for each latency set. In these figures, "w/o local" means result when the proposed scheme was not applied to the programs, and "w/ local" means result when the proposed scheme was applied. Lat20, lat40 and lat80 are evaluated latency sets described in Table 1 respectively.

In Figure 3, results of "w/ local" show scalable speedup, for example 6.87 times(lat20), 5.92 times(lat40) and 5.02 times(lat80) when using 8PEs. On the other hand, results of "w/o local" don't show scalable speedup. Comparing results of "w/ local" and "w/o local" when using 8PEs, "w/ local" is 1.76 times faster than "w/o local"(lat20), 2.25 times(lat40) and 2.62 times(lat80). This is because "w/o local" configurations are suffered from CSM access contentions with the increase of the number of processors, while "w/ local" scheme can use LDM effectively.

In Figure 4, results of "w/ local" show scalable speedup, for example 6.55 times(lat20), 5.79 times(lat40) and 4.84 times(lat80) when using 8PEs. On the other hand, results of "w/o local" don't show scalable speedup. Comparing results of "w/ local" and "w/o local" when using 8PEs, "w/ local" is 2.26 times faster than "w/o local"(lat20), 3.49 times(lat40) and 4.96 times(lat80). This is because "w/o local" configurations are suffered from CSM access contensions with the increase of the number of processors, while "w/ local" scheme can use LDM effectively.

5 Conclusions

This paper has proposed local memory management on OSCAR CMP for data localization. The proposed scheme is implemented on OSCAR Fortran multigrain parallelizing compiler.

Its performance is evaluated on OSCAR CMP architecture using Swim and Tomcatv from the SPEC fp 95. The result of evaluation show us that speedups of Swim and Tomcatv for 8 PEs were 6.87 times and 6.55 times respectively against sequential execution time.

A part of this research has been supported by STARC "Automatic Parallelizing Compiler Cooperative Single Chip Multiprocessor", Grants-in-Aid for JSPS Fellows(# 1501202), the project of Advanced Research Institute for Science and Engineering, Waseda University "Automatic Parallelizing Compiler cooperative Chip Multiprocessor" and JSPS Grants-in-Aid for Young Scientists(B)(# 15700074).

References

[1] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *Technical White Paper*, Oct 2001.

[2] Kevin Krewell. UltraSPARC IV mirrors predecessor. In *Microprocessor Report*, Nov. 2003.

[3] K. Kimura, T. Kodaka, M. Obata, and H. Kasahara. Multigrain parallel processing on compiler cooperative oscar chip multiprocessor architecture. In *IEICE Trans. ELECTRON.*, Apr 2003.

[4] H. Kasahara, M. Obata, K. Ishizaka, K. Kimura, H. Kaminaga, H. Nakano, K. Nagasawa, A. Murai, H. Itagaki, and Shirako J. Multigrain automatic parallelization in japanese millenium project it21 advanced parallelizing compiler. In *Proc. of IEEE PARELEC*, Sep 2002.

[5] A. Yoshida, K. Koshizuka, M. Okamoto, and Kasahara H. A data-localization scheme among loops for each layer in hierarchical coarse grain parallel processing. *Trans. of IPSJ*, 40(5):2054–2063, 1999.

[6] H. Kasahara. *Parallel Processing Technology.* CORONA PUBLISHING CO., LTD., 1991.