J Supercomput (2011) 57: 51–64
DOI 10.1007/s11227-011-0561-0

Multi-CMP system with data communication on the fly

Marek Tudruj · Lukasz Masko · Miroslaw Thor

Published online: 4 February 2011
© The Author(s) 2011. This article is published with open access at Springerlink.com

Abstract The paper concerns new communication solutions for hierarchical Chip Multiprocessor (CMP) systems composed of many CMP modules interconnected by a global data exchange network. New architectural solutions for internal module data communication are presented in the presence of hierarchical data caches in CMP modules. Inside CMP modules, dynamic shared memory core clusters are organized around L1–L2 data cache busses. Such clusters enable a group-oriented data communication based on reads on the fly to L1 banks of data present on the busses by many cores at a time. Dynamic switching of cores between such L1–L2 busses is done with porting data in core's L1 caches. Together with data reads on the fly, it provides a very efficient intercluster "communication on the fly," especially useful for transfers of strongly shared data. It provides fast cache to cache group data transmissions and eliminates standard transactions based on shared memory in the system. Comparative experimental results based on automatic scheduling of program data flow graphs and execution in a simulator of the proposed architecture evaluate the assumed architectural solutions. The multi-CMP system structure is assessed while taking into account technological limitations of the size of the single CMP module.

M. Tudruj (✉) · L. Masko
Institute of Computer Science of the Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland
e-mail: tudruj@ipipan.waw.pl

L. Masko
e-mail: masko@ipipan.waw.pl

M. Tudruj
Polish–Japanese Institute of Information Technology, ul. Koszykowa 86, 02-008 Warsaw, Poland

M. Thor
Telemark University College, Hallvard Eikas plass, 3800 Bo i Telemark, Norway
e-mail: miroslaw.thor@hit.no

Keywords Modular CMP systems · Program execution control · Dynamic shared memory clusters · Communication on the fly

1 Introduction

Cluster-based approach is an important solution, which can improve communication efficiency in parallel shared memory systems. Distribution of memory traffic among many Shared Memory Processor (SMP) clusters has been extensively used to alleviate the memory bus saturation problem in large shared memory processor systems. In standard implementations, the size of the SMP clusters is fixed, which can decrease parallel program execution efficiency due to discrepancy between system structure and program needs. A more ambitious strategy assumes modifiable cluster sizes, which can adjust system structure to real needs of an application problem leading to a better use of communication resources.

Parallel shared memory systems implemented in Chip Multiprocessor (CMP) technology attract currently much of the systems architects' attention [6, 11]. Data communication efficiency in such systems is determined to the large extent by data exchange latency of transfers between processor data caches, which are the actual sources and destinations of shared data. Therefore, in current CMP systems, internal communication between on-chip cores and elements of the hierarchy of data caches is an important problem considered in research papers [1–6, 11, 13]. Especially, dynamically reconfigurable communication of strongly shared data between processor cores and their data caches deserve high designers' interest; however, no sufficient attention in current papers has been paid to efficient group communication on chip.

Cluster-based approach is currently in common use, but up-to now this approach has not been ported to the area of the internal network structures of CMP systems. This paper presents new solutions in this respect and proposes a new data communication method for parallel numerical computations in dynamic cluster-based shared memory system architecture for the CMP technology. The new method is called communication on the fly [14–16]. It enables dynamic creation of temporary shared memory core clusters (SMPs), which provide means for fast group transmission of shared data between cores, more exactly between their L1 data caches. In this paper, the initial solution based on a single level data caches [14], has been extended to multilevel caches. It improves time efficiency of the proposed group communication since it concerns now faster L1, L2 data cache memories. Dynamic SMP clusters are organized using local data networks, which connect L1 data caches with L2 cache banks (L1–L2 packetized data busses) to enable data reads on the fly of shared data. Communication on the fly is a new intercluster data exchange method. Its main idea consists in switching processor cores with some relevant data in their L1 caches to new L1–L2 bus-based clusters to pass there new data by a group-like read on the fly to L1 caches of many cores at a time. In this way, classic data exchange by deposing data in a shared memory to be next read by many processes or threads is replaced by processor core switching and a single group transaction on a data bus. The core switching is done during program execution for program defined time interval, longer than for a single memory transaction, which distinguishes this solution from a standard shared

memory access. It strongly reduces contention on memory bus lines and eliminates many separate data transactions concerning the same data. Although current implementations use multihop packet switching on-chip networks [2, 13], it is the data bus which provides the data and address observability features necessary for this type of group communication.

The core switching and communication on the fly can be automatically embedded in the application program code by special scheduling algorithms [8, 9] in a way, which corresponds to particular program needs. In this respect, the applied communication method shows features of dynamically reconfigurable embedded system, in which internal structures of CMP modules are adjusted to program requirements. Such systems enable accelerated execution of typical time consuming numerical computations in problem-oriented libraries.

Simulation experiments have shown the potential of communication on the fly for fine grain parallel programs [14–16]. However, current CMP technology limitations imposed by power dissipation, wire delays, signal cross talks and silicon area space make that the physical system structure should be based on many cooperating CMP modules, containing a limited number of cores and L2 cache banks, rather than on a single large CMP module. The CMP system modular architecture also comes from characteristics of shared memory data busses, which for a limited number of bus customers behave in an acceptable manner, not worse than other types of networks [1], while showing valuable features relevant for the described efficient group data transactions. Most of the cited experiments concerned standard parallel matrix multiplication based on recursive data decomposition into quarters. In this paper, we evaluate the speedups of scheduled general layered parallel program graphs executed by simulation in the proposed architecture for different assumptions concerning the number of dynamic CMP modules used, the speed of cache memory busses and the speed of the global network.

The paper is composed of three main parts. In the first part, new system general architectural features are discussed. Next, an extended macro data flow graph representation is explained, which has been used for simulation experiments. In the last part, efficiency of the parallel programs in the proposed architecture is evaluated by simulation experiments with program graphs execution.

2 General system architecture

In this paper, we present a new version of a general system architecture based on dynamic SMP clustering, which is tailored to be implemented in the chip multiprocessor technology. We investigate a hierarchical modular structure of many CMP modules connected by a central global network, Fig. 1. A CMP module contains a number of processor cores C with directly used L1 data caches, which are connected to shared L2 data cache memory banks using CMPs local L1–L2 data networks. Each CMP module has directly accessible fragment of the shared data main memory placed in the address space common for all modules. The global network provides data exchange between shared data memory modules of CMP modules at the cost of higher data transfer latency compared to internal transfers of data inside particular CMPs.

Fig. 1 The general system structure

The internal structure of a CMP module is shown in Fig. 2. Processor cores C_i (each connected with its local L1 data cache and shared L2 cache banks) can be dynamically interconnected in dynamic clusters through local communication network composed of L2 busses (local busses, L1–L2 busses). Additional instruction caches are also provided, but discussing them is beyond the scope of this paper. A new collective data communication in clusters of cores can be organized inside CMP modules. It consists in dynamic switching of cores with data cache contents between SMP clusters organized on L2 bank busses. This method converts data transmissions through memory and/or some global network into dynamic cluster reconfiguration with data transfers performed directly between data caches. Such a conversion enables, multiple parallel reads of data by many processors to their L1 data caches from L2 bank busses (reads on the fly, similar to cache injection [10]) while a processor writes data from its L1 cache to the cluster L2 bank. It is based on address snooping of L2 bank bus by a core Bus Request Controller (BRC) in which read on the fly requests are queued when issued from the application programs. Reads on the fly combined with processor switching, called "communication on the fly," provide a very fast way of data exchange between processor clusters. During reads on the fly, the data copies can be written with new memory addresses if are to be later subject to modification, thus avoiding cache coherence problems.

Local busses used for local data transmission inside CMP modules, provide a very efficient intra-cluster communication on the fly especially useful for transfers of shared data. Operations on data include: data prefetch from L2 cache to a core L1, write from L1 to L2, read on the fly from a L2 bank bus to many L1s in parallel, core switching between L2 busses (core clusters), data pre-fetches from shared memory modules to L2 banks, data updates of memory modules from L2 banks, inter-CMP module data exchange accomplished as transfers between shared data memory

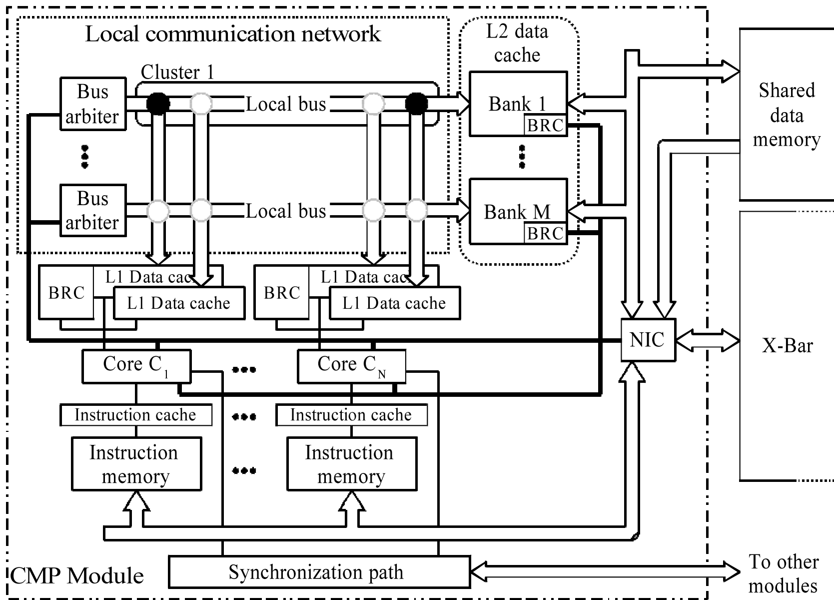

Fig. 2 Internal structure of a CMP module

modules. Before prefetches of data to L1, the corresponding data blocks must be preloaded to L2 from the respective memory modules in a programmed way. Current task computing results are sent from L1 to the L2 data cache module only after a task completes.

The global data exchange is performed between shared memory modules attached to CMPs. It is done under control of the Network Interface Controller (NIC), which collects global data transfer requests from the CMP cores. The NIC creates connections in the global network (which can be a crossbar switch or a multi-layer Closs connection network). The shared memory modules are dual-ported, so that it is possible to perform simultaneously a data read/write on the “shared memory-L2” bus and a read from the shared data memory for a global network transmission.

The local communication networks inside CMP modules are provided with very advanced data exchange features. They consist in multiported data caches L1 that enable parallel prefetching of arguments of subsequent numerical operations and many communications on the fly performed at a time for a processor core. The multiported data cache L1 provides much better functionality and performance than a cache shared by many processors based on address interleaving among multiple cache banks since it ensures many parallel copies of shared data available to processors.

The L1 and L2 data caches are used in this architecture in many cases as “scratchpad memories” rather than with the functionality of classic cache memory. It is because the cache-controlled macro data flow program execution paradigm enforces strong data prefetching to L1 caches, which can be also extended on the L2 data cache behavior. Following special data cache functionality, the system ensures L1 and L2 data caches synchronization in the context of processor core switching and reads on the fly. If a processor core is switched from one L2 module to another and is to write

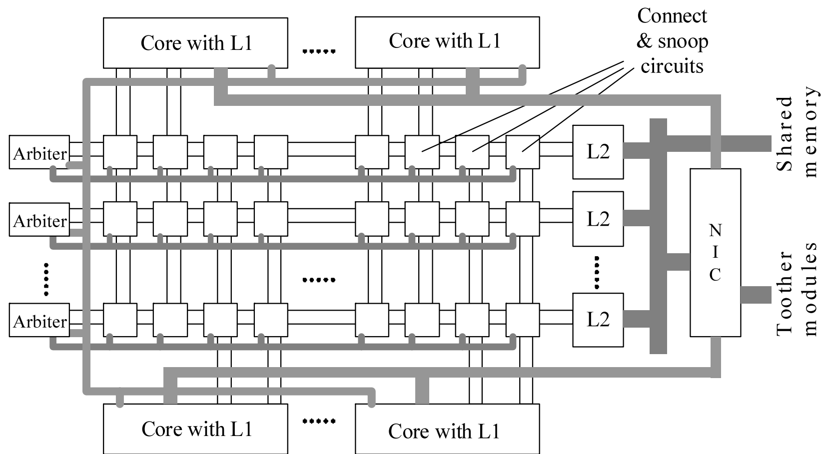

Fig. 3 Floor-plan of the proposed CMP module

some data from its L1 to this new L2 module (for instance to enable other processors to read these data through reads on the fly or simply to do a cache block flush), a respective new line in the target L2 data cache must be provided. This operation is not performed in a standard manner (by transmitting proper data from the shared memory), but instead, just before the L2 write operation, an empty line in L2 is generated together with the address tag and a special validity mask. The mask controls in terms of L1 blocks which data will be moved in by the considered data transfer on the fly from the L1–L2 bus. When necessary, the operating memory will be updated using only the valid parts of the L2 lines. Similar actions are performed in L1 caches, when data are prefetched into L1 under new addresses to comply with the assumed single-assignment principle, which eliminates consistency problems for multiple copies of modified shared data (data read for subsequent modification are stored under new addresses). This imposes new dummy lines in L1 data caches provided with similar control fields. Only on L1 to L2 flushing or reads on the fly through the L2 busses the corresponding L2 dummy lines will be generated (a lazy synchronization is used).

The floor-plan of the proposed CMP module is shown in Fig. 3. The active elements of a CMP module (cores, L1 caches, L2 caches, bus arbiters, NIC controller) can be placed around the local communication network which implements switched connections and reads on the fly with address snooping.

3 Extended macro-data-flow graph representation

An application program for the proposed architecture is built following the structure of its Macro Data Flow Graph (MDFG). The macro nodes in this graph are sequences of program instructions meant for execution in a single core. They are defined by a programmer according to the policy he uses for program parallelization. The macro nodes are atomic entities executed in parallel in the system. To describe activities of processor cores in dynamic clusters, we propose an Extended Macro Data Flow

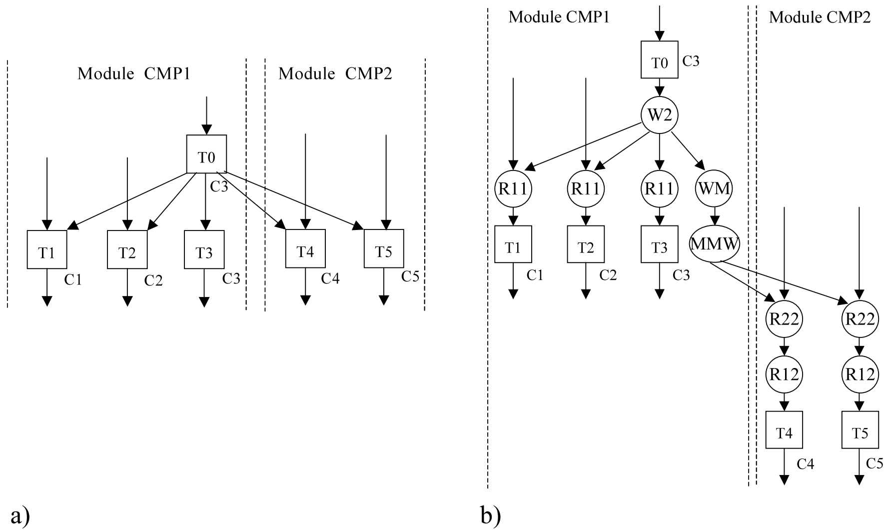

Fig. 4 (*Right*) Figure presents an EMDFG (**b**) of a simple program whose MDFG is shown in (**a**) meant for execution

Graph notation (EMDFG) in which special kinds of nodes are used in the program graph: data prefetch (read) nodes to the data cache L1 from L2 (R1), data write nodes from processor's L1 data cache to L2 (W2), write of some data from L2 to the shared memory of the CMP module (WM), L2 data cache prefetch from shared memory (RL2), write from the shared memory of a CMP module to the memory of another CMP module (MMW), read from a distant CMP module shared memory to the memory of a CMP module (R2), processor switch nodes between L2 data cache banks, barriers used to synchronize reads on the fly with the write which supplies data (B), intracluster (local) bus arbiter nodes, the intercluster global bus arbiter node. The read and write nodes weights correspond to volumes of data measured in L1 data cache blocks. EMDFGs can be generated manually by programmers but it is complicated and error prone. What we recommend is that EMDFGs are generated automatically by a scheduling program [8, 9] whose input is a program MDFG and which takes into account the parameters of the target system with the proposed architecture. A scheduled EMDFG will be then transformed into an executive code in which special instructions, representing the described above special elements of the extended graph, will be automatically inserted.

Figure 4 (right) presents an EMDFG of a simple the program MDFG shown in Fig. 4 (left) meant for execution in a system without processor core switching nor reads on the fly. In these graphs, program macro nodes (T0...T5) are assigned to processor cores (C1...C5) in CMP modules (CMP1, CMP2). There are the following new nodes in the graph: W2—write of data from core's C3 data cache L1 to L2 inside CMP1, R11—data prefetch from L2 to L1 in CMP1, WM—write of some data from the L2 to the shared memory of CMP1, MMW—write from the shared memory of a CMP1 to the memory of CMP2, R22—prefetch of data from shared memory to L2 banks in CMP2, R12—prefetch of data from L2 to L1 in CMP2.

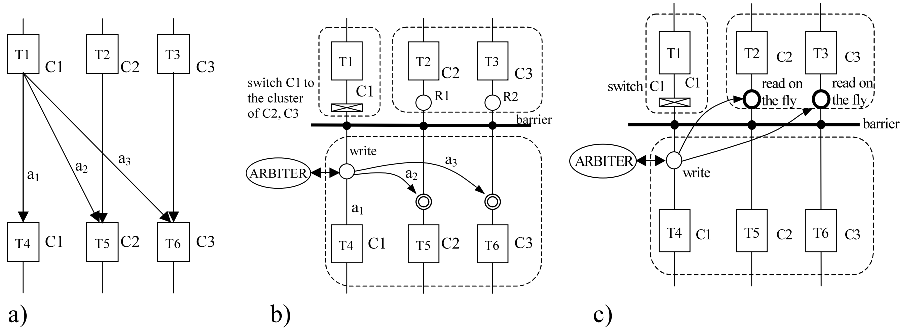

Fig. 5 EMDFG with communication on the fly: (**a**) standard MDFG, (**b**) equivalent EMDFG with communication on the fly, (**c**) simplified EMDFG notation

An example of the extended macro data flow program graph for a program with communication on the fly is shown in Fig. 5. The graph is composed of program nodes T1–T6. Nodes T4, T5, T6 receive data through communication on the fly from node T1 in a system composed of cores C1, C2, C3 placed in the same CMP module. To represent a synchronized read on the fly, a read on the fly node can be decomposed into two nodes, (Fig. 5b): the read on the fly request deposing in a BRC (R1, R2) performed before the barrier working for the "write" node and the read on the fly execution (double circle node) performed when the barrier is reached. A special node marked in Fig. 5 as a crossed rectangle (with L2 module specification), represents switching of a core to a new dynamic cluster. The write node execution time depends on the bus access acknowledge sent from the bus arbiter. In communication on the fly, processor cores can read parts of data on the bus, which differentiates this solution from a standard multicast or broadcast. In read on the fly instructions, delimiters are specified which determine read starting addresses (a_i in Fig. 5) and data volumes (fractions of the total data volume sent on the bus can be captured). Since in a cache memory environment the transactions on the memory busses concern entire cache blocks, the delimiters are expressed in terms of the cache blocks.

A section in a program graph (denoted by a dotted oval rectangle in Fig. 5) is a subgraph executed by a fixed subset of processor cores connected to the same local L2 bus, i.e., belonging to the same cluster. Cores are notified about newly starting sections to activate all program parallel threads specified for them. By the mechanism of sections, communication (data read and write) requests sending to BRCs can be adjusted to current sections, i.e., to the use of the right busses and the composition of core clusters.

4 Experimental results

The efficiency of parallel program execution in the proposed architecture has been verified by experiments with artificially generated program graphs. Since the assumed system is strongly modular and the communication on the fly requires strict synchronization of the participating cores, the structures of the examined program graphs

were selected to be of modular and synchronous character. It means that the graphs have layered structures in which some roughly regular subgraphs with intensive communication can be identified. These graphs have similar structure and granularity to graphs of many regular parallel numerical programs like parallel matrix multiplication or parallel FFT computations.

The graphs have been automatically scheduled for execution in the proposed architecture with and without communication on the fly for different assumptions regarding time and structural parameters of the executive system. The number and structure of the CMP modules, the speed of L1–L2 busses and the latency of the global networks (a crossbar switch) were considered. Next, the scheduled graphs were symbolically performed using a simulator, which is cycle accurate if the communication control is concerned, using the approach similar to [12]. The simulator was written by the authors in C++ language. It determined program execution times based on which parallel speedups were computed against execution on a single processor.

Programs have been structured by a heuristic scheduling algorithm [9] based on the use of the notion of moldable tasks [7]. This algorithm assigns tasks to processor cores and communication to L1–L2 busses in CMP modules in a way which reduces application program execution time. A moldable task is a part of a program, whose structure can be tailored for execution on different numbers of processors, assuming that for each number of processors a best execution time can be determined. The moldable task approach enables a hierarchical approach to program scheduling in which first characteristics of moldable tasks, as building program components, are found and then an optimal program structuring of these building elements is designed. The macro data flow graph representation of programs is used for this optimization. The algorithm first builds a graph of admissible moldable tasks (it was assumed not wider than 4 nodes) for a given program and then moldable tasks are scheduled for a given number of CMP modules with particular characteristics. To find moldable tasks, the algorithm schedules program tasks for execution inside the CMP modules with communication on the fly. Next, it schedules the global communication between the CMP modules. As a result, a scheduled program moldable task graph is produced in which computation and internal moldable task communication are assigned to resources inside CMP modules and global communication between the tasks is assigned to links of the global network.

The general structure of experimental program graphs is shown in Fig. 6. The graphs have the following parameters—the number of levels: 10; number of intensive communication subgraphs: 8; the width of a level in a subgraph: 4; the total graph width: 32, the computing node weight: 100; the communication edge weight: 20–30, 100–150; the node output degree: 1–2, 3–4. Up to 25 additional edges of inter-subgraph data transfers have been added to the graphs. Both the internal data communication in CMP modules and global inter-module communication were placed at the application program levels (neither communication libraries nor the operating systems support were used). The width of the moldable tasks generated by the task scheduling algorithm was in all cases limited to 4. Thus, the moldable tasks could be embedded inside single CMP modules.

The relation between the speed of processor cores and local communication speed (i.e. the frequency of the L1–L2 bus) was assumed 2:1, 4:1, 8:1 (2:1 is equivalent to

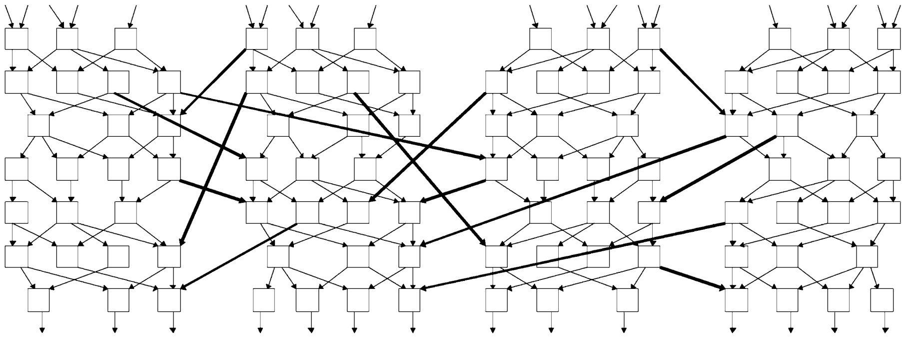

Fig. 6 Experimental program graph structure

1.6 GFLOPS processor with L1 cache co-operating with L2 cache via 800 MHz bus). The latency of the global network was assumed to be 2, 8 or 16 times bigger than that of the L1–L2 bus. The following cache memory bus and global network speeds configurations were examined: 8_2, 8_8, 8_16, 16_2, 16_8, 16_16. The number of cores in the system was always equal to 32. However, the cores were embedded in 1, 2, 4, and 8 CMP modules with communication on the fly, containing 32, 16, 8, and 4 cores, respectively. Additionally, a system with 32 single-core processors interconnected by a global network without communication on the fly was considered. Therefore, the following configurations of the numbers of CMP modules and core numbers in modules were considered: 1_32, 2_16, 4_8, 8_4, 32_1. The total system configuration description (for example 4_8_8_2) is composed of the number of CMP modules (4), the number of cores in a CMP module (8), the L1–L2 bus speed coefficient (8) and the global network speed coefficient (2).

In figures below, the average parallel execution speedup against sequential execution of programs in different system configurations as explained above is presented in whose macro data flow graphs the computational node degree was 3–4 or 1–2 and the data transaction volume was 100–150 or 20–30. The local communication time (inside a CMP module) was measured as a product of the data volume and the L1–L2 bus speed coefficient and the global communication time was determined as the product of the local communication time and the global network speed coefficient. The experiments correspond to programs with intensive (frequent) local data communication whose latency is much larger than the computation time of nodes. We can observe that for fast global networks the obtained speedup was the highest for the single large CMP module. This is due to communication on the fly and the absence of the global network.

However, the technology reasons enforce the use of multiple smaller CMP modules to avoid large values of the signal propagation time on long bus wires and the diversity of this time depending on the physical placement of the bus customer. For high global network speed, Fig. 7, the speedup of execution on 2 CMP modules for dense communication with big data volumes was close to the results for the system with a single CMP module. For 4 and 8 CMP modules with a top global network speed, the speedup decreased from 23 to 20 and 17, respectively, due to global communication. It shows that the use of smaller CMP modules for not very frequent

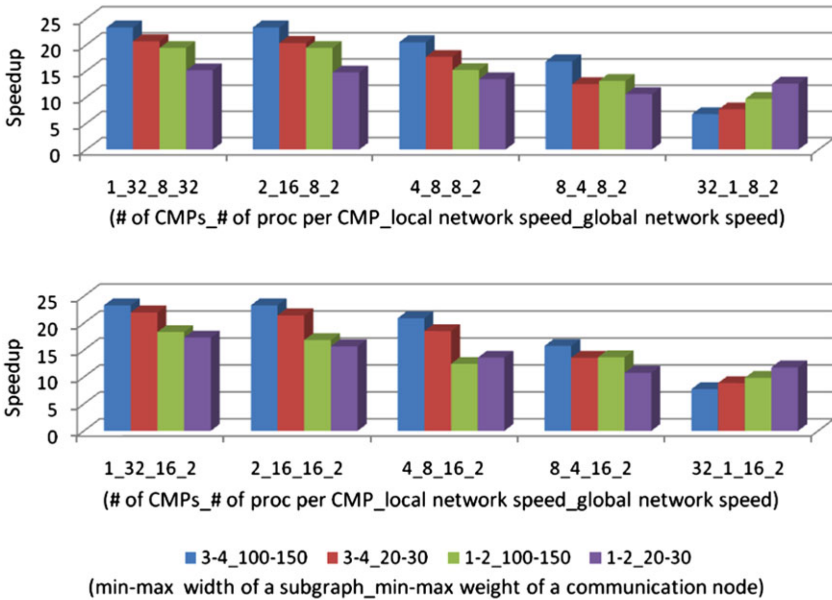

Fig. 7 Parallel speedup for fast global network

global communication in programs can be advantageous. For less intensive communication (node degree 1–2 and small communication volume), the speedup decrease was from 15 to 11. The system configuration with 32 single-core processors connected exclusively by the global network (configurations 32_1_8_2 and 32_1_16_2, where all data transfers were global) has provided the small speedup which increased for the programs with less intensive communication.

Figure 8 presents program execution speedups when the global network was very slow (8 times slower than the L1–L2 bus) programs contained a much less communication, however, for large data volumes. Due to weak global communication, the speedup for dense communication for 2, and 4 CMP modules is still not so much lower than for a single module, but it degrades below 15 when communication is not intensive nor large, which gives the parallelization efficiency below 0.5. This efficiency is above 0.5 for voluminous and dense communication with 4 and 8 CMP modules and small L1–L2 bus speed, due to the positive influence of the communication on the fly.

In all discussed cases of the relatively weak global communication, the best speedups were obtained for small number of CMP modules and for dense and voluminous communication. Based on other research not shown in this work, we can state that more dense global communication gives smaller speedups for larger numbers of CMP modules and low global network speed. Therefore, the speed and the architecture of the global network are crucial in such cases.

Figures 9 and 10 present improvements of speedups due to application of communication on the fly corresponding to large and small communication volumes (20–30 and 100–150), for different configurations of system parameters. In these experiments, a list scheduling algorithm combined with a genetic algorithm was used [9]. It

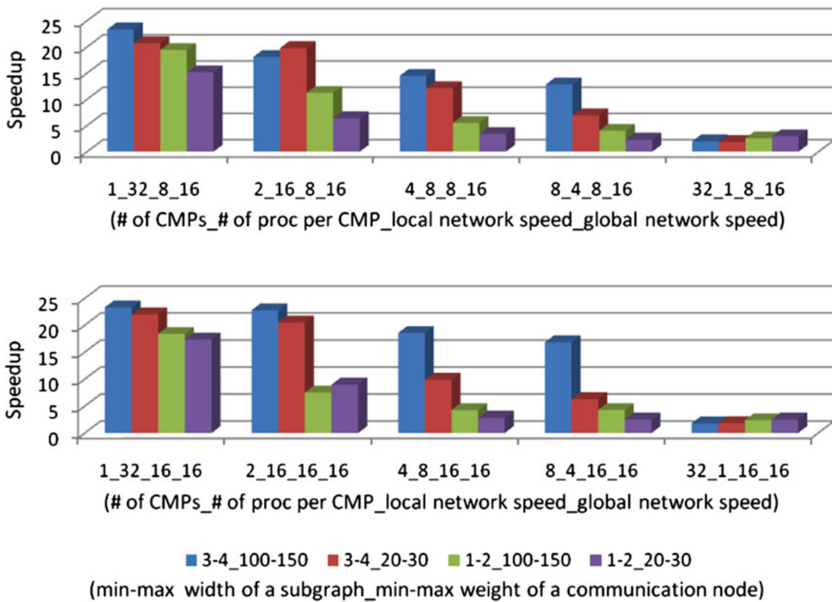

Fig. 8 Parallel speedup for very slow global network

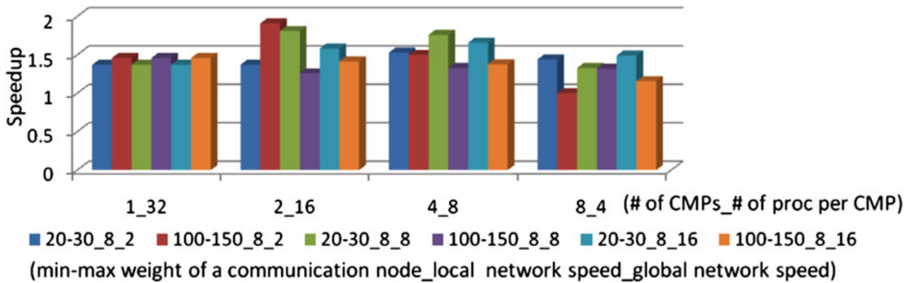

Fig. 9 Parallel speedup improvement due to communication on the fly for fast L2 busses

enables better examination of the influence of communication on the fly on parallel speedup since the structuring of the program is simpler than in the case of moldable tasks approach. We can see that communication on the fly gives an average speedup improvement of 1.44 for faster L1–L2 busses (8 times slower than the core-L1 communication) and 1.22 for slower L1–L2 busses (16 times slower than the processor core-L1 bus), comparing the scheduling without communication on the fly.

5 Conclusions

In the paper, we have presented and examined system architecture based on multiple CMP modules interconnected by a global network with a special new feature of the communication on the fly inside the CMP modules. Communication on the fly can be

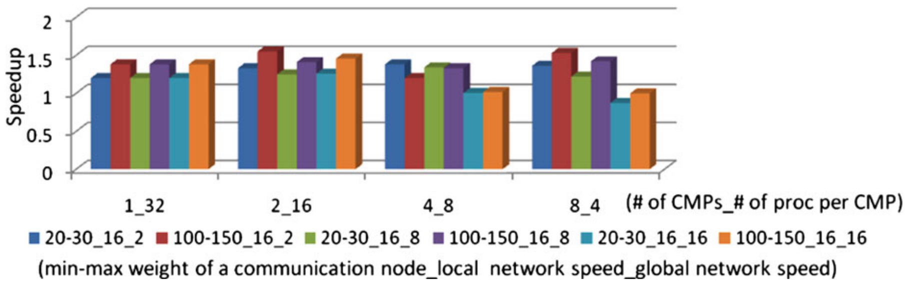

Fig. 10 Parallel speedup improvement due to communication on the fly for slow L2 busses

an important data exchange mechanism for execution of computational algorithms in which strong data sharing appears among parallel fragments of programs. It enables strong reduction of data traffic on busses which lead from processor cores to shared L2 data caches and main memory modules. This type of communication should be embedded in special CMP modules meant for execution of numerical fragments of parallel programs. Its positive impact on efficiency of parallel computations grows if the degrees of data sharing and the synchronous layer-based data processing in parallel programs are higher. The use of many smaller size CMP modules interconnected by a global network can be enforced by technology limitations. The speedups of execution of computational programs with layered graph structures composed of communicating subgraphs with relatively low inter-subgraph communication were examined. We can see that for such assumptions this architecture behaves in a satisfactory way for fast global networks and for a relatively small number of CMP modules; so, the use of several CMP modules for programs as examined, provides not much worse results than the use of a single large CMP module. With multiple CMP modules applied for programs with more intensive global communication, high global network speed and proper architectural properties of the network to reduce the global communication influence are of big importance.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

1. Adriahantenaina et al (2003) SPIN: a scalable, packet switched on-chip micro-network. In: Proceedings of the design, automation and test in Europe, DATE'03, Munich, March, vol 2, pp 2070–2081
2. Dally W, Towles B (2001) Route packets, not wires: on-chip interconnection networks. In: 38-th DAC, Las Vegas, June, pp 684–689
3. Hossain H et al (2008) Improving support for locality and fine-grain sharing in chip multiprocesors. In: PACT'08, Toronto. ACM Press, New York, pp 155–165
4. Hsu L et al (2005) Exploring the cache design space for large scale CMPs. SIGARCH Comput Archit News 33(4), November
5. Huh J (2007) A NUCA substrate for flexible CMP cache sharing. IEEE Trans Parallel Distrib Syst 18(8):1028–1040
6. Kundu S, Peh LS (2007) On-chip interconnects for multicores. In: IEEE MICRO, Sept–Oct, pp 3–5

7. Lepere R, Trystram D, Woeginger GJ (2001) Approximation algorithms for scheduling malleable tasks under precedence constraints. In: 9th annual European symposium on algorithms. LNCS, vol 2161. Springer, Berlin, pp 146–157
8. Masko L, Tudruj M (2008) Task scheduling for SoC-based dynamic SMP clusters with communication on the fly. In: 7th int symp on parallel and distributed computing, ISPDC 2008, Krakow, pp 99–106
9. Masko L et al (2005) Scheduling moldable tasks for dynamic SMP clusters in SoC technology, parallel processing and applied mathematics. In: PPAM 2005, Poznań, Poland, Sept 2005. LNCS, vol 3911. Springer, Berlin, pp 879–887
10. Milenkovic, Milutinovic V (2000) Cache injection: a novel technique for tolerating memory latency in bus-based SMPs. In: Proceedings of the euro-par. LNCS, vol 1900. Springer, Berlin
11. Owens JD et al (2007) Research challenges for on-chip interconnection networks. In: IEEE MICRO, Sept–Oct, pp 96–108
12. Parisha S, Dutt N, Ben-Romdhane M (2008) Fast exploration of bus-based communication architectures at CCATB abstraction. ACM Trans Embed Comput Syst 7(2)
13. Terry TY et al (2004) Packetization and routing analysis of on-chip multiprocessor networks. J Systems Archit 50:81–104
14. Tudruj M, Masko L (2004) Dynamic SMP clusters with communication on the fly in NoC technology for very fine grain computations. In: 3rd int symp on parallel and distributed computing, ISPDC 2004, Cork, July, pp 97–104
15. Tudruj M, Masko L (2005) Towards massively parallel computations based on dynamic SMP clusters with communication on the fly. In: 4th int symp on parallel and distributed computing, ISPDC 2005, Lille, France. IEEE CS Press, Los Alamitos, pp 155–162
16. Tudruj M, Masko L (2006) Fast matrix multiplication in dynamic SMP clusters with communication on the fly in systems on chip technology. In: PARELEC 2006, September. IEEE CS Press, Los Alamitos, pp 77–82