Mapping Algorithms for Multiprocessor Tasks on Multi-core Clusters

Jörg Dümmler
Chemnitz University of Technology
Department of Computer Science
djo@cs.tu-chemnitz.de

Thomas Rauber
Bayreuth University
Angewandte Informatik II
rauber@uni-bayreuth.de

Gudula Rünger
Chemnitz University of Technology
Department of Computer Science
ruenger@cs.tu-chemnitz.de

Abstract

In this paper, we explore the use of hierarchically structured multiprocessor tasks (M-tasks) for programming multi-core cluster systems. These systems often have hierarchically structured interconnection networks combining different computing resources, starting with the interconnect within multi-core processors up to the interconnection network combining nodes of the cluster or supercomputer. M-task programs can support the effective use of the computing resources by adapting the task structure of the program to the hierarchical organization of the cluster system and by exploiting the available data parallelism within the M-tasks. In particular, we consider different mapping algorithms for M-tasks and investigate the resulting efficiency and scalability. We present experimental results for different application programs and different multi-core systems.

1. Introduction

The M-task programming model has originally been proposed to combine the benefits of task and data parallelism. An M-task program is hierarchically subdivided into a set of M-tasks each working on a different part of the application (task parallelism). An M-task is a parallel task which can run on an arbitrary number of cores or processors, leading to varying execution times. We distinguish basic M-tasks and composed M-tasks. Basic M-tasks are implemented using an SPMD programming style, e.g. MPI or OpenMP, and comprise the actual computations to be performed (data parallelism). Composed M-tasks are used to describe the interactions between basic M-tasks or other composed M-tasks. Within a composed M-task, other M-tasks can be activated for execution. The advantage of this approach is to increase the available degree of parallelism and to restrict communication within M-tasks to subsets of the available processors or cores, thus reducing the communication overhead and increasing scalability.

An application program given as an M-task specification offers several possibilities for a parallel execution, differing in the order in which the M-tasks are executed and the subsets of processors or cores assigned to each M-task. On different parallel architectures different versions of the M-task program might be the most efficient and scalable ones. To find an optimal M-task program version is an NP-complete problem which is usually solved by heuristic algorithms. In this article, we extend the M-task approach to heterogeneous multi-core systems and propose mapping algorithms which are aware of the heterogeneity of multi-core systems. The contributions of this article include:

- to explore the use of the M-task programming model for multi-core systems;
- to suggest mapping algorithms for assigning M-tasks to specific processor cores and to explore the resulting differences in execution time;
- to investigate the resulting scalability for several benchmarks from the area of solvers for ordinary differential equation (ODEs) and from the NAS parallel benchmarks.

The investigations on dual-core and quad-core systems show significant differences in the performance of the proposed mappings depending on the communication pattern of the application. The lowest execution times are usually achieved by using a consecutive mapping that places processes belonging to the same M-task onto the same cluster node. Additionally, we show that the performance of ODE solvers can be further improved by exploiting special communication patterns based on an orthogonal arrangement of the processes and the combination of MPI and OpenMP into hybrid execution schemes.

The rest of the paper is organized as follows. Section 2 describes the multiprocessor task programming model. Section 3 describes the mapping algorithms for M-tasks. Section 4 presents a detailed evaluation of the mapping strategies for different recent parallel systems. Section 5 discusses related work and Section 6 concludes and discusses future work.

2. M-Task Programming

An M-task program is built up from a set of M-tasks cooperating with each other. The coordination between M-tasks can be based on control or data dependencies. M-tasks have a set of input parameters and produce a set of output parameters; both are visible to other M-tasks. A data dependence between M-tasks M_1 and M_2 arises if M_1 produces output data required as an input for M_2.

A data dependence may lead to a data re-distribution operation if M-task M_1 provides its output data in a different distribution or on a different set of processors or cores than it is expected by M-task M_2. These data dependencies or control dependencies emerging from the structure of the application lead to precedence constraints between M-tasks. Precedence constraints restrict the possible execution order of the M-tasks. If M-tasks M_1 and M_2 are connected by a precedence constraint the execution of M_1 must have been finished and all required data re-distribution operations must have been carried out before the execution of M_2 can be started. For independent M-tasks a concurrent execution on disjoint subsets of the available processors as well as an execution one after another are possible.

Due to the precedence constraints between M-tasks, M-task programs can be represented by a graph structure $G_M = (V, E)$, where the set of nodes V consists of the M-tasks of a program and edges $e \in E$ connect different M-tasks M_1 and M_2 if there is a precedence constraint between M_1 and M_2. A precedence constraint can be a data or control dependence from M_1 to M_2. Examples for graph structures G_M are macro dataflow graphs in the Paradigm compiler[3] or SP-graphs within the TwoL model[13].

For the parallel execution of the M-task program represented by G_M there exist several execution schemes differing in

i) the number of cores assigned to each M-task and
ii) the execution order for independent M-tasks, i.e. for M-tasks $M_1, \ldots, M_k \in V$ that are not connected by a path within G_M (scheduling M-tasks).

For heterogeneous parallel platforms there is an additional degree of freedom in the parallel execution strategy, which is the assignment of specific processors (or processor cores) to M-tasks (mapping M-tasks). The heterogeneity of the execution platform might be caused by different processors (or processor cores), different nodes, or different interconnections between cores, processors, and nodes.

3. Mapping Algorithms

Determining a suitable execution scheme of an M-task program for a specific heterogeneous multi-core machine requires several steps: scheduling the execution order of the M-tasks, determining the number of cores assigned to each M-task, and mapping the M-tasks to specific cores. If the cores assigned to an M-task do not have a shared address space, the data distributions of the input and output parameters also have to be fixed. In the following, we concentrate on the mapping decisions and explore different mappings for M-tasks to specific processor cores.

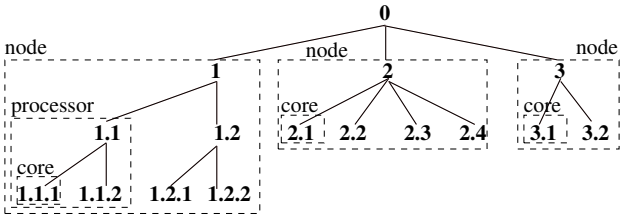

Figure 1. Use of Dewey notation for the representation of heterogeneous platforms.

3.1. Architecture Model

In this paper we consider multi-core systems with a hierarchy of computing resources as a special case of a heterogeneous platform. We assume cores of the same type but with different interconnections between (i) cores of the same processor, (ii) processors of the same node, and (iii) nodes of a partition or the entire machine. The architecture can be represented as a tree-structure with cores C as leaves, processors P as intermediate nodes being a parent for cores, computing nodes N as intermediate nodes combining processors, and partitions or the entire machine A as root node.

For a unique identification of intermediate levels and leaf cores of the architecture tree, we use the Dewey notation [7]. Each node n in the tree gets a label $l(n)$ consisting of digits separated by dots; the label uniquely describes the path from the root to the specific node. The root gets the label $l(n) = 0$. The children of the root get the labels $1, \ldots, k$ from left to right where k is the number of children. The labels for the nodes in the following levels are defined recursively. The label $l(n)$ for a node n starts with the label from the parent node m and concatenates a digit i if n is the root of the subtree i, $i = 1, \ldots, k$ separated by a dot: $l(n) = l(m).i$. This notation is suitable for describing heterogeneous hierarchical multi-core systems, see Fig. 1.

3.2. Mapping

The mapping of M-tasks to processor cores has to take the task structure and the architectural structure into account. In the following, we assume that the execution order and the number of executing cores for each M-task has been determined in a scheduling step. For the scheduling, a set of homogeneous symbolic cores is used which has to be mapped to physical cores in the mapping step. For the

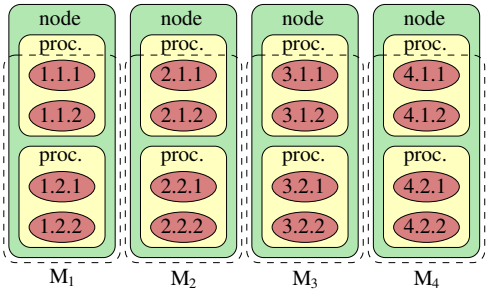

Figure 2. Example for a consecutive mapping of M-tasks $M_i, i = 1, \ldots, 4$ each requiring 4 symbolic cores on a platform with 4 nodes consisting of 2 dual-core processors.

mapping, we consider the situation that g M-tasks are executed concurrently to each other using a group partitioning $G = (G_1, \ldots, G_g)$ of symbolic processor cores such that group G_i with $|G_i| = g_i$ executes M-task M_i. The mapping can be described by a function

$$F : \{G_1, \ldots, G_g\} \rightarrow 2^{\mathcal{C}}$$

where $\mathcal{C}$ denotes the set of physical cores; F maps a symbolic group G_i to a physical group $F(G_i)$ and it is $F(G_i) \cap F(G_j) = \emptyset$ for $i \neq j$. Moreover $|G_i| = |F(G_i)|$, i.e., each symbolic group is mapped to a physical group of the same size.

In the following, we propose several mappings mainly differing in the strategies how symbolic cores are mapped to physical cores of the parallel machine. Since the underlying programming model like MPI or OpenMP also influences performance and communication times, the specific choice of the mapping function also has to take this into consideration. For the definition of the mappings, we assume a platform consisting of $|N|$ identical nodes, each containing $|P|$ processors with $|C|$ cores. For each proposed mapping, we define a sequence of physical cores

$$s_1, s_2, \ldots s_m \quad \text{with} \quad m = |N| * |P| * |C|.$$

Each physical core appears exactly once in this sequence. The mapping function F assigns the symbolic cores of a group $G_i, i = 1, \ldots, g$ to consecutive physical cores in this sequence, i.e.

$$F(G_i) = \left\{ s_j, s_{j+1}, \ldots, s_{j+|G_i|-1} \mid j = 1 + \sum_{k=1}^{i-1} |G_k| \right\}.$$

Node-oriented consecutive mapping: For this mapping, symbolic cores are mapped consecutively to physical cores to obtain a node-oriented use of the physical cores. If a group of symbolic cores is larger than the number of physical cores per node of the architecture, several nodes are used such that each node is used only for one group. Otherwise, more than one group may be mapped to one node. This mapping tries to minimize the number of groups that are mapped to each node of the architecture. Fig. 2 shows

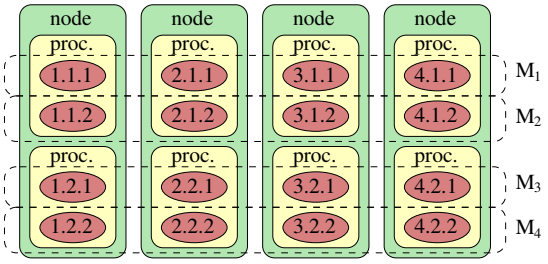

Figure 3. Example for a scattered mapping of M-tasks $M_i, i = 1, \ldots, 4$ each requiring 4 symbolic cores on a platform with 4 nodes consisting of 2 dual-core processors.

an example. This mapping should be beneficial if communication within nodes is faster than communication between nodes for the specific target architecture and intra M-task communication outweighs inter M-task communication.

In this mapping, the physical cores are ordered such that cores of the same node are adjacent, i.e. the sequence of physical cores is

$$1.1.1, ..., 1.1.|C|, 1.2.1, ..., 1.|P|.|C|, 2.1.1, ..., |N|.|P|.|C|.$$

The label $l(s_i) = n_i.p_i.c_i$ of an arbitrary core $s_i, 1 \leq i \leq m$ in this sequence can be computed by

$$n_i = 1 + \left\lfloor \frac{i-1}{|P| * |C|} \right\rfloor, \quad p_i = 1 + \left(\left\lfloor \frac{i-1}{|C|} \right\rfloor \bmod |P| \right),$$
$$\text{and} \quad c_i = 1 + ((i-1) \bmod |C|).$$

Scattered core-level mapping: For this mapping, the physical cores for a specific group of symbolic cores are selected such that cores are used whose Dewey notation end with the same digit, i.e., corresponding cores of different nodes are used. Fig. 3 illustrates this mapping. If a group contains less symbolic cores than nodes, one physical core of each node is used for the mapping. This ensures an equal participation of the nodes in the communication performed during the execution of the M-tasks of the group.

The sequence of physical cores for this mapping is defined as

$$1.1.1,...,|N|.1.1, 1.1.2,...,|N|.1.|C|, 1.2.1,...,|N|.|P|.|C|.$$

The label of a core s_i in this sequence $l(s_i) = n_i.p_i.c_i$ is given by

$$n_i = 1 + ((i-1) \bmod |N|), \quad p_i = 1 + \left\lfloor \frac{\lfloor (i-1)/|N| \rfloor}{|P|} \right\rfloor,$$
$$\text{and} \quad c_i = 1 + (\lfloor \tfrac{i-1}{|N|} \rfloor \bmod |P|).$$

Mixed core-node level mapping: Node-oriented and core-oriented mapping strategies can be mixed, e.g. to adapt to the ratio of intra vs. inter M-task communication. Mixed mappings can be described using the parameter $d, 1 \leq d \leq |P| * |C|$ denoting the number of consecutive physical cores of a node used to execute an M-task. If the

number of symbolic cores assigned to an M-task is greater than d, multiple nodes are used for the execution. The scattered mapping results for $d = 1$ and $d = |P| * |C|$ leads to the consecutive mapping. The sequence of physical cores is given by

$$1.1.1, ..., 1.(1 + \frac{d-1}{|C|}).(1 + ((d-1) \bmod |C|)), 2.1.1,$$
$$..., |N|.(1 + \frac{d-1}{|C|}).(1 + ((d-1) \bmod |C|)), ...,$$
$$1.(1 + \frac{2d-1}{|C|}).(1 + ((2d-1) \bmod |C|)), ..., |N|.|P|.|C|.$$

The label $l(s_i) = n_i.p_i.c_i$ of the ith core in the sequence can be computed by

$$n_i = 1 + \left(\left\lfloor \frac{i-1}{d} \right\rfloor \bmod |N| \right), p_i = 1 + \left\lfloor \frac{r}{|C|} \right\rfloor,$$
$$c_i = 1 + (r \bmod |C|)$$
$$\text{with } r = \left\lfloor \frac{\lfloor (i-1)/d \rfloor}{|N|} \right\rfloor * d + ((i-1) \bmod d).$$

4. Experimental Results and Evaluation

In this section, we investigate the resulting execution time for the different mappings for different execution platforms and benchmark applications. We also consider communication optimizations between M-tasks and hybrid realizations of M-tasks with MPI and OpenMP.

4.1. Experimental Setup

4.1.1. Hardware Description

For the benchmark tests, a variety of platforms is used. The Chemnitz High Performance Linux Cluster (CHiC) is built up of 538 nodes consisting of two AMD Opteron 2218 dual-core processors with a clock rate of 2.6 GHz and a peak performance of 5.2 GFlops/s per core. The communication between the nodes is performed with the MVAPICH 1.0beta MPI library over a 10 GBit/s infiniband network.

The SGI Altix system consists of 19 partitions. The benchmarks are executed inside a partition containing 128 nodes, each equipped with two Intel Itanium2 Montecito dual-core processors. The processors are clocked at 1.6 GHz and achieve a peak performance of 6.4 GFlops/s per core. Each node has two links to the NUMAlink 4 interconnection network offering a bidirectional bandwidth of 6.4 GByte/s. The employed MPI library is SGI MPT 1.17.

The Xeon cluster consists of 2 nodes with 2 Intel Xeon E5345 'Clovertown' quad-core processors each. The processors run at 2.33 GHz and have a peak performance of 9.33 GFlops/s per core. An infiniband network with a bandwidth of 10 GBit/s connects the nodes and the MVAPICH2 1.0beta library supplies the MPI functionality.

4.1.2. Benchmark Description

A class of applications that benefit from the M-task programming model are solution methods for ordinary differential equations (ODEs) that compute a fixed number K of independent stage vectors in each time step. The computation of each stage vector can be represented by an M-task. Examples for explicit solvers for non-stiff ODEs are Iterated Runge-Kutta (IRK) methods[16] and Parallel Adams-Bashforth(PAB) methods; implicit solvers are Parallel Adams-Moulton(PAM) methods for non-stiff ODEs and Diagonal-Implicitly Iterated Runge-Kutta (DIIRK) methods for stiff ODEs. The combination of the PAB and PAM methods in a predictor-corrector scheme results in an implicit ODE solver (PABM)[15].

Each time step of the IRK method computes m fixed point iteration steps with an implicit Runge-Kutta corrector. Each fixed point iteration step involves an intra M-task multi-broadcast (MPI_Allgatherv()) and a global data re-distribution between M-tasks. At the end of each time step the new approximation is computed using global communication. Similar to IRK, the DIIRK method computes m corrector steps each requiring a global data exchange. In each corrector step of DIIRK, a system of non-linear equations is solved using Newton's method with Gaussian elimination leading to multiple intra M-task communication operations. Compared to IRK, DIIRK requires more computations, and intra M-task communication is more important. In the PABM method, the computation of the stage vectors involves multiple intra M-task multi-broadcast operations. Global communication is only required once at the end of each time step. For the benchmarks, a sparse ODE system (2D Brusselator discretization) and a dense ODE system (resulting from a Schrödinger-Poisson system) are used.

Another class of applications that can benefit from the M-task programming model are solvers for flow equations that operate on a set of meshes (also called zones). A single time step involves independent computation within each zone followed by a border exchange between overlapping zones. The NAS parallel benchmark multi-zone version (NPB-MZ)[17] provides solvers for discretized versions of the unsteady, compressible Navier-Stokes equations that operate on multiple zones. The fine grain parallelism within the zones is exploited using shared memory OpenMP programming; the coarse grain parallelism between the zones is realized using message passing with MPI. For the purpose of this article we consider a modified version of the Lower-Upper Symmetric Gauss-Seidel multi-zone (LU-MZ) benchmark which uses MPI for both levels of parallelism. This has the advantage that several nodes of a distributed memory platform can operate on the same zone. There are 16 zones in this benchmark that can be represented by 16 independent M-tasks.

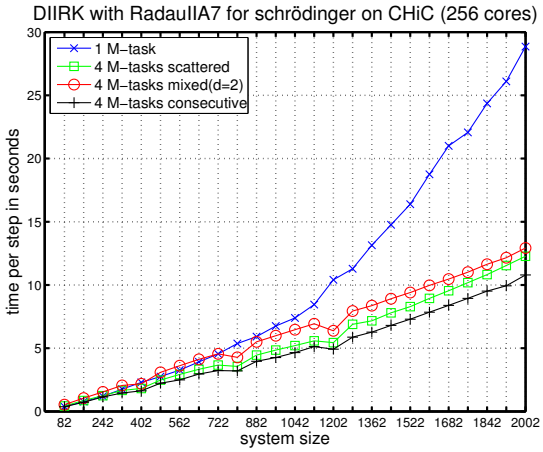

Figure 4. Runtimes of DIIRK using the four stage RadauIIA7 method on CHiC.

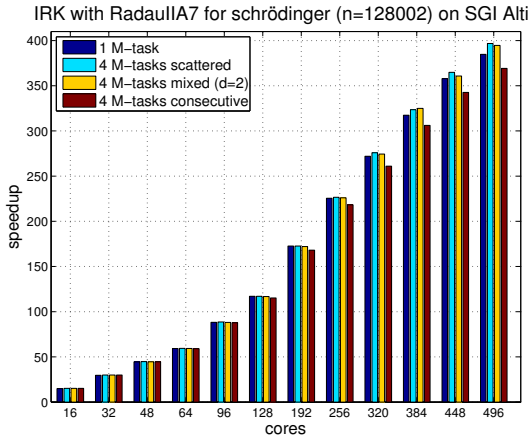

Figure 5. Speedups of IRK using the four stage RadauIIA7 method on SGI Altix.

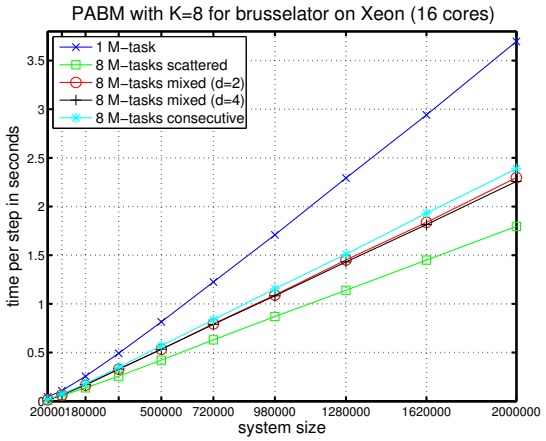

Figure 6. Runtimes of PABM using $K = 8$ stages on the Xeon cluster.

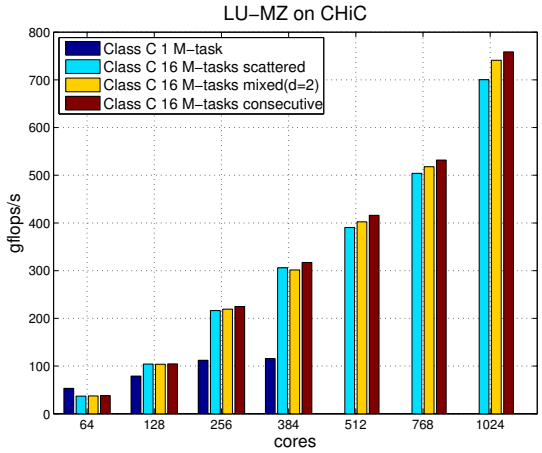

Figure 7. Performance of the LU-MZ benchmark on CHiC.

4.2. Evaluation of Mappings to Cores

First, we compare a pure data parallel execution scheme with a task parallel execution. At each point in time, the data parallel scheme executes one M-task using all $|\mathcal{C}|$ available cores and the task parallel version executes the maximum number K of independent M-tasks on K disjoint groups consisting of $\lfloor |\mathcal{C}|/K \rfloor$ or $\lceil |\mathcal{C}|/K \rceil$ cores.

Fig. 4 shows the execution times of a time step of DIIRK using four stages on the CHiC cluster. Task parallelism results in much lower runtimes because intra M-task communication can be restricted to groups of cores. Data transfers are faster within a cluster node of the CHiC leading to a clear performance benefit of the consecutive mapping.

Fig. 5 shows the speedups of IRK using four stages on the SGI Altix. Intra M-task communication is less important in IRK and therefore data parallelism is competitive. For the task parallel versions, a scattered mapping delivers the best results. This mapping leads to the placement of processes with the same group rank on the same node. Therefore, the exchange of the stage vectors between each fixed point iteration is carried out within a node.

The execution times for different mappings of PABM on the Xeon cluster are shown in Fig. 6. Compared to IRK, PABM requires considerably more communication within the M-tasks. On the other hand, in this configuration $K = 8$ M-tasks are executed using 2 cores each leading to a small communication overhead for each M-task compared to redistributions between M-tasks. Therefore, for configurations with many M-tasks that are executed by only few cores the lowest execution times are achieved by a scattered mapping.

The total GFlops per second achieved by the LU-MZ benchmark are shown in Fig. 7 for the CHiC cluster and in Fig. 8 for the SGI Altix. Problem classes 'C' with a global mesh size of $480 \times 320 \times 28$ and 'D' with a global mesh size of $1632 \times 1216 \times 34$ are used. The data parallel version of class 'C' can only be executed for up to 448

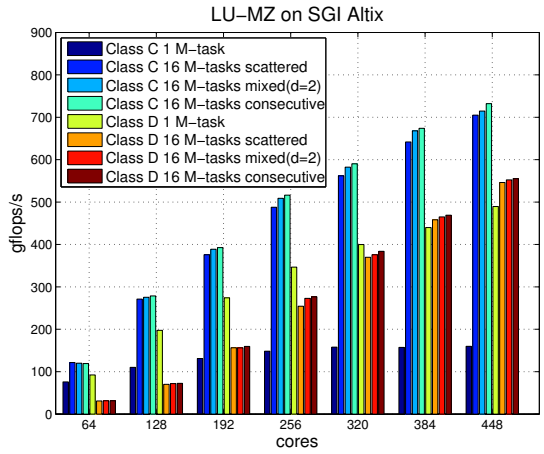

Figure 8. Performance of the LU-MZ benchmark on SGI Altix.

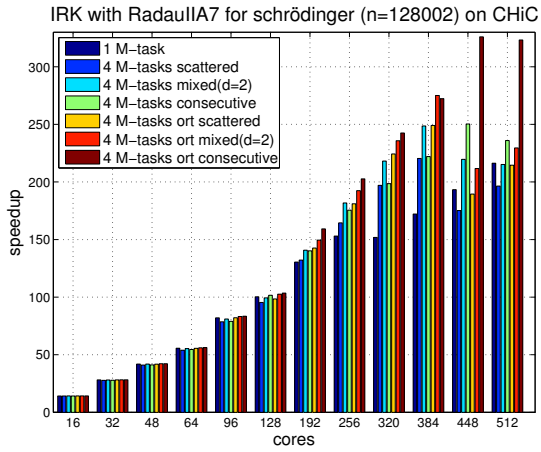

Figure 9. Speedups of IRK using the four stage RadauIIA7 method on CHiC.

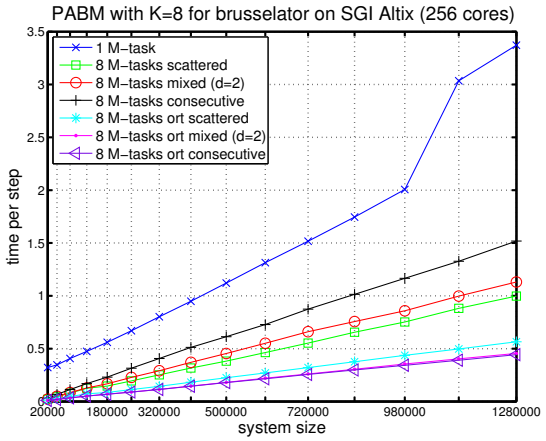

Figure 10. Runtimes of PABM using $K = 8$ stages on SGI Altix.

cores because a minimum amount of data is required for each process. For a low number of cores, pure data parallelism leads to better results because a data exchange between zones is not required. But on a high number of cores the communication within the zones becomes more important because the amount of data assigned to each process becomes smaller. The node consecutive mapping leads to the best performance on both platforms. For class 'D' the computation to communication ratio is much higher leading to smaller differences between the program versions.

4.3. Optimizing Communication Between Tasks

The inter M-task communication within the considered ODE solvers can be optimized by exploiting a special communication pattern based on an orthogonal arrangement of the MPI processes[12]. In these orthogonal program versions the data exchange between M-tasks is carried out by concurrent multi-broadcast operations.

Fig. 9 compares the speedups for data parallel, standard task parallel and orthogonal task parallel execution schemes of IRK on the CHiC cluster. For a low number of cores, most of the execution time is spent within computations leading to an almost equal performance of all program versions. But for a high number of cores, a suitable mapping is required for a good performance. The results show that a mixed or a consecutive mapping perform best for the standard task parallel version whereas the consecutive mapping clearly leads to the highest speedups for the orthogonal version.

Similar to IRK, the execution times of PABM can be reduced significantly by employing orthogonal communication. Fig. 10 shows the execution times on the SGI Altix. The scattered mapping achieves the lowest execution times for the standard task parallel version. For the orthogonal program version a consecutive and a mixed mapping achieve the best results because less time is spent in data exchanges between M-tasks.

4.4. MPI Tasks vs. OpenMP Threads

An adaption to the hardware characteristics of clusters of SMPs can be achieved by combining message passing with MPI and thread programming with OpenMP into hybrid programming models. In this section, we examine the performance of hybrid realizations of M-task programs. The upper level parallelism between M-tasks is realized by MPI communication and for the lower level parallelism within M-tasks hybrid MPI+OpenMP implementations are used. Multiple processes of the same M-task have to be mapped on the same cluster node to make use of the OpenMP parallelism. Therefore, a suitable mapping strategy is required. In the following, we focus on a consecutive mapping for both, pure MPI and hybrid implementations.

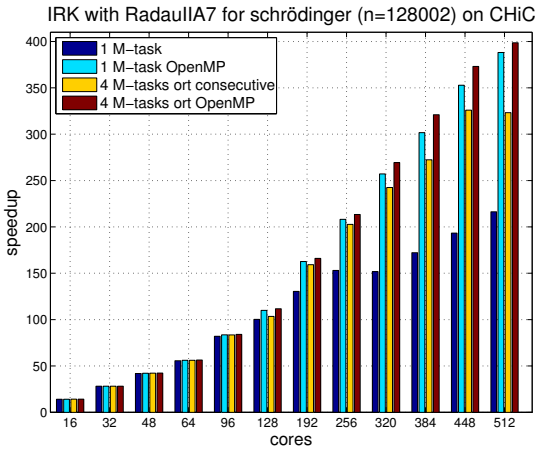

Figure 11. Speedups of pure MPI and hybrid MPI+OpenMP realizations of IRK with the four stage RadauIIA7 method on CHiC.

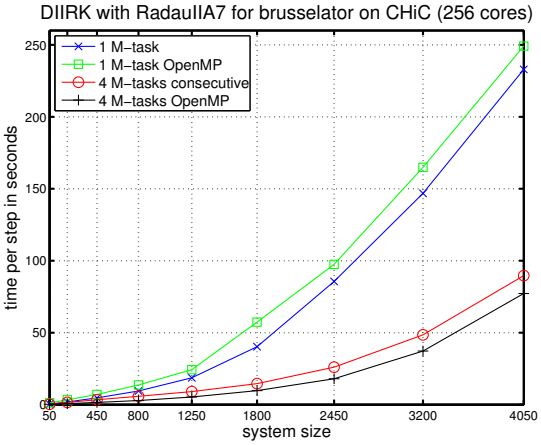

Figure 12. Runtimes of pure MPI and hybrid MPI+OpenMP versions of DIIRK on CHiC.

A comparison of the achieved speedups of IRK with four stages on the CHiC cluster using four OpenMP threads per cluster node is shown in Fig. 11. The hybrid execution scheme for the data parallel version leads to considerable higher speedups compared to a pure MPI realization. The main source of this improvement is the reduction of the number of participating MPI processes in global communication operations. The best results for IRK are obtained by using orthogonal communication between nodes and OpenMP intra-node.

Fig. 12 shows the execution times of a time step of DIIRK with four stages on CHiC. The hybrid execution leads to a slow-down for the data parallel version caused by program parts that require a frequent synchronization, e.g. the pivoting in the Gaussian elimination. For the task parallel version, the hybrid execution scheme clearly outperforms its pure MPI counterpart.

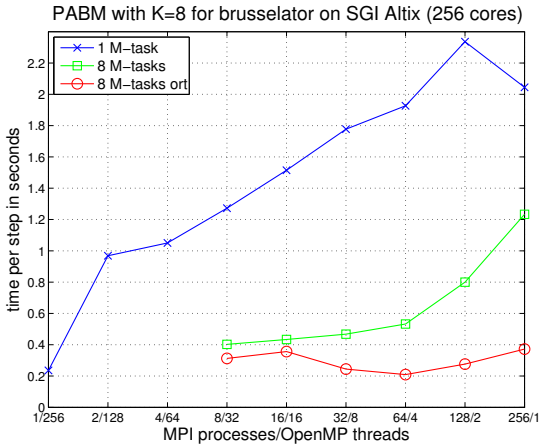

Figure 13. Runtimes of different combinations of MPI processes and OpenMP threads of PABM with $K = 8$ stages on SGI Altix.

The SGI Altix has a distributed shared memory architecture that allows the use of OpenMP threads across different nodes. Therefore, many different combinations of MPI processes and OpenMP threads are possible. Fig. 13 shows a comparison of the execution times of PABM with eight stages on 256 cores of the SGI Altix. At least eight MPI processes are required for task parallelism. Using the maximum possible number of OpenMP threads leads to the best results for both, the data parallel and the standard task parallel version. For the orthogonal version, the lowest execution times are achieved by using 64 MPI processes and 4 OpenMP threads.

5. Related Work

Several research groups have proposed models and programming environments to support mixed task and data parallel executions with the goal to obtain parallel programs with faster execution time and better scalability properties, see [2, 14] for an overview and see [4] for a detailed investigation of the benefits of combining task and data parallelism.

Scheduling algorithms for computing an appropriate mix of task and data parallelism for M-tasks programs on homogeneous platforms are presented in [18, 19, 11]. These algorithms can be combined with the proposed mappings to provide a better utilization of the computing resources of multi-core clusters. An overview of M-task scheduling algorithms for heterogeneous target platforms is given in [9]. These approaches are targeted to large cluster-of-clusters systems and restrict the execution of single M-tasks to a homogeneous sub-cluster. Our benchmark results show that M-tasks have to be executed across multiple nodes of multi-core clusters to obtain high speedups. Therefore, these

heterogeneous scheduling algorithms are not suitable for multi-core clusters.

The mapping of tiles of distributed multi-dimensional arrays with nearest neighbor communication on nodes of an SMP cluster has been studied in [5]. Mapping heuristics for data parallel applications on physical processors have been presented in [10, 1, 6]. Profiling is used to obtain the communication requirements of the target application and a mapping heuristic selects an optimized placement of the processes based on the communication performance of the target platform. Mixed task and data parallel applications and dependencies between processes are not explicitly taken into account. A heuristic for mapping a set of independent tasks consisting of a fixed number of threads to multi-core processors has been introduced in [8].

6. Conclusions and Future Work

In this paper, we have proposed several mapping strategies for M-task programs on multi-core systems. Investigations on different platforms show a significant difference in the performance for different mappings and for different execution schemes. The optimal mapping usually depends on the ratio of communication within M-tasks and data exchanges between M-tasks. M-tasks with a high communication requirement benefit from a mapping on consecutive cores of the same cluster node. Exploiting orthogonal communication patterns and hybrid MPI+OpenMP leads to a further increase of the performance. As example applications, several large programs from scientific computing have been considered. Future work includes the investigation of cost models to predict the impact of the proposed mappings and providing an automatic selection of the best mapping strategy.

Acknowledgement

We thank the Leibniz Rechenzentrum München for providing access to the SGI Altix system.

References

[1] T. Agarwal, A. Sharma, and L. V. Kalé. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proc. of the 20th Intl. Parallel and Distributed Processing Symposium (IPDPS 2006)*. IEEE, 2006.

[2] H. Bal and M. Haines. Approaches for Integrating Task and Data Parallelism. *IEEE Concurrency*, 6(3):74–84, July-August 1998.

[3] P. Banerjee, J. Chandy, M. Gupta, E. Hodge, J. Holm, A. Lain, D. Palermo, S. Ramaswamy, and E. Su. The Paradigm Compiler for Distributed-Memory Multicomputers. *IEEE Computer*, 28(10):37–47, 1995.

[4] S. Chakrabarti, J. Demmel, and K. Yelick. Modeling the benefits of mixed data and task parallelism. In *Symposium on Parallel Algorithms and Architecture*, pages 74–83, 1995.

[5] D. Chavarría-Miranda, J. Nieplocha, and V. Tipparaju. Topology-aware tile mapping for clusters of smps. In *CF '06: Proceedings of the 3rd conference on Computing frontiers*, pages 383–392, New York, NY, USA, 2006. ACM.

[6] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn. Mpipp: an automatic profile-guided parallel process placement toolset for smp clusters and multiclusters. In *ICS '06: Proc. of the 20th Int. Conf. on Supercomputing*, pages 353–360, New York, NY, USA, 2006. ACM.

[7] D. E. Knuth. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*. Addison Wesley, Reading, Mass., 1975. 2nd edition, 2nd printing.

[8] Y. Liu, X. Zhang, H. Li, and D. Qian. Allocating Tasks in Multi-core Processor based Parallel Systems. In *2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC 2007)*, 2007.

[9] T. N'takpé, F. Suter, and H. Casanova. A Comparison of Scheduling Approaches for Mixed-Parallel Applications on Heterogeneous Platforms. In *Proc. of the 6th Int. Symp. on Parallel and Distributed Computing*. IEEE, 2007.

[10] J. M. Orduna, F. Silla, and J. Duato. On the development of a communication-aware task mapping technique. *J. Syst. Archit.*, 50(4):207–220, 2004.

[11] A. Radulescu, C. Nicolescu, A. van Gemund, and P. Jonker. CPR: Mixed Task and Data Parallel Scheduling for Distributed Systems. In *IPDPS '01: Proc. of the 15th Int. Parallel & Distributed Processing Symp.* IEEE, 2001.

[12] T. Rauber, R. Reilein, and G. Rünger. Group-SPMD Programming with Orthogonal Processor Groups. *Concurrency and Computation: Practice and Experience, Special Issue on Compilers for Parallel Computers*, 16(2-3):173–195, 2004.

[13] T. Rauber and G. Rünger. A Transformation Approach to Derive Efficient Parallel Implementations. *IEEE Transactions on Software Engineering*, 26(4):315–339, 2000.

[14] D. Skillicorn and D. Talia. Models and languages for parallel computation. *ACM Comp. Surveys*, 30(2):123–169, 1998.

[15] P. van der Houwen and E. Messina. Parallel Adams Methods. *J. of Comp. and App. Mathematics*, 101:153–165, 1999.

[16] P. J. van der Houwen and B. P. Sommeijer. Iterated runge-kutta methods on parallel computers. *SIAM J. Sci. Stat. Comput.*, 12(5):1000–1028, 1991.

[17] R. van der Wijngaart and H. Jin. The NAS Parallel Benchmarks, Multi-Zone Versions. Technical Report NAS-03-010, NASA Ames Research Center, 2003.

[18] N. Vydyanathan, S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. An integrated approach for processor allocation and scheduling of mixed-parallel applications. In *Proc. of the 2006 Int. Conf. on Parallel Processing (ICPP'06)*. IEEE, 2006.

[19] N. Vydyanathan, S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. Locality conscious processor allocation and scheduling for mixed parallel applications. In *Proc. of the 2006 IEEE Int. Conf. on Cluster Computing*. IEEE, 2006.