Scalable Computing with Parallel Tasks

Jörg Dümmelr  Thomas Rauber  Gudula Rünger

TU Chemnitz  Universität Bayreuth  TU Chemnitz

MTAGS Workshop at SC’09, Nov. 16, 2009
Introduction and Motivation

- Parallel tasks (M-tasks) can be used to structure parallel programs in a flexible way by capturing the available degree of parallelism.
  → **M-task programming model** originally designed for large *cluster systems*;
- M-tasks can be used to combine the benefits of *task and data parallelism*.
  - advantage: increase the available *degree of parallelism*;
    restrict communication within M-tasks to *subsets* of processors;
  → reduce the *communication overhead* and *increase scalability*.
- We extend the M-task approach to *heterogeneous multi-core systems* and propose suitable *scheduling* and *mapping algorithms*.
- We investigate the *resulting scalability* for several *application programs*;
  solvers for *ordinary differential equations (ODEs)*;
  *NAS parallel benchmarks*;
M-task Programming Approach

- An M-task program is **hierarchically subdivided** into a set of M-tasks each executing a **different part** of the application (**task parallelism**).
- An **M-task** is a parallel task which can be executed on an **arbitrary number of processors** (malleable task).
- Independent tasks can be executed concurrently by **disjoint processor groups**;
- We distinguish **basic M-tasks** and **composed M-tasks**.
- **Basic M-tasks** are formulated in **SPMD-style** and can be computed internally in a **data-parallel or SPMD way** (or threads) **OpenMP or MPI** $\rightsquigarrow$ specific mixed parallel execution;
- **Composed M-tasks** are used to describe the **interactions** between basic M-tasks or other composed M-tasks.
- Each M-task can be activated at **different program points** with a varying number of processors.
Execution of M-task Programs

- The coordination between M-tasks can be based on control or data dependencies;
  → precedence constraints between M-tasks;
- M-tasks have a set of input parameters and produce a set of output parameters; → data dependencies;
- Data dependencies may make data re-distribution operations between cooperating M-tasks necessary;
- Dependencies restrict the execution order of M-tasks;
  → dependency graph;
- there exist several execution schemes for the parallel execution of an M-task program differing in:
  - the number of cores assigned to each M-task;
  - the execution order for independent M-tasks (scheduling);
  - the assignment of specific cores to M-tasks (mapping);
Outline

1 Introduction and Motivation

2 Scheduling and Mapping

3 Experimental Evaluation

4 Conclusions
Scheduling of M-tasks

- **Scheduling algorithm** determines the execution order of the M-tasks within a task graph (using symbolic cores).
- Mapping algorithm defines corresponding sets of physical cores.
- **Cost model** for scheduling: for \( q \) cores and mapping pattern \( mp \), the cost of an M-task \( M \) is defined as
  \[
  T(M, q, mp) = T_{\text{comp}}(M, 1)/q + T_{\text{comm}}(M, q, mp)
  \]
- Additionally, **re-distribution costs** between cooperating M-tasks \( M_1 \) and \( M_2 \) may occur:
  \[
  T_{\text{Re}}(M_1, M_2, q_1, q_2, mp_1, mp_2)
  \]
- **Layer-based scheduling**: partition the M-task graph into layers of independent M-tasks and schedule the layers one after another.
Scheduling Algorithm

for each layer $W = \{M_1, \ldots, M_k\}$ {
    let $P(W)$ be the set of symbolic cores for $W$;
    let $p(W)$ be the number of symbolic cores for $W$;
    $T_{min} = \sum_{i=1}^{k} T_{symb}(M_i, p(W))$;
    for each $g \in \text{(set of divisors of } k)$ do {
        partition $P(W)$ into $g$ subsets $G = \{G_1, \ldots, G_g\}$ of size
        $p_g = p(W)/g$;
        sort $\{M_1, \ldots, M_k\}$ such that $T_{symb}(M_1, p_g) \geq \ldots \geq T_{symb}(M_k, p_g)$;
        for ($j = 1, \ldots, k$)
            assign $M_j$ to $G_l$ with the smallest accumulated execution time;
        $T_{act}(g) = \max_{1 \leq j \leq g}$ accumulated execution time of $G_j$;
        if ($T_{act}(g) < T_{min}$) then $T_{min} = T_{act}(g)$;
    }
}
Scheduling Algorithm – post-adjustment

- The layers of an M-task graph are scheduled one after another.
- **scheduling within a single layer:**
  
  several reasonable numbers of processor subsets of equal size are tested for each layer – select the best

- **post-adjustment of the group sizes:**
  
  adjust the size $g_l$ of each subset $G_l$ to the accumulated computational work of the M-tasks assigned to the groups:

  $$
g_l = \text{round} \left( \frac{T_{\text{seq}}(G_l)}{\sum_{j=1}^g T_{\text{seq}}(G_j) \cdot p(W)} \right).
  $$

- additional improvement for **distributed address space:**
  
  select same subsets for **neighboring layers** with dependencies;
  
  use **orthogonal communication** between subsets;
Scheduling algorithm defines group partitioning $G = (G_1, \ldots, G_g)$ with symbolic cores for each layer of the task graph; mapping determines corresponding sets of physical cores;
Node-oriented consecutive mapping

- Symbolic cores are mapped **consecutively** to physical cores; if necessary, **several nodes** are used for a subset;

- beneficial if **intra-node communication** is faster than **inter-node communication**
Scattered core-level mapping

- corresponding cores of different nodes are used for a subset;
- equal participation of nodes in intra-M-task communication;

- beneficial if inter-M-task-communication outweights intra-M-task communication;
Mixed core-node level mapping

- mix of **core-oriented** and **node-oriented** mapping strategies;
- $d$ consecutive physical cores of a node are used for an M-task;
- **illustration** for $d = 2$

---

$d$ can be adapted to the ratio of intra- and inter-M-task communication
Outline

1. Introduction and Motivation
2. Scheduling and Mapping
3. Experimental Evaluation
4. Conclusions
Experimental evaluation – platforms

- **CHIC cluster**: 2 AMD Opteron dual-core per node; IBM MPI; 538 nodes; MVAPICH library;
- **SGI Altix LRZ Munich**: 2 Itanium 2 dual-core per node; 128 nodes per partition, SGI MPT library;
- **JuRoPa cluster**, Forschungszentrum Jülich: 2 Intel Xeon “Nehalem“ quad-core processors per node; 2208 nodes; QDR Infiniband network; ParaStation MPI library v5.0 and the Intel Compiler 11.0.
Experimental evaluation – applications

- **numerical solution methods for systems of ODEs:**
  - Iterated Runge-Kutta (IRK) methods;
  - Parallel Adams Bashforth (PAB) methods;
  - Parallel Adams-Moulton (PAM) methods;
  - Diagonal-Implicitly Iterated Runge-Kutta (DIIRK);

**sparse ODE systems:** spatial discretization of a 2D reaction-diffusion equation (Brusselator)

**dense ODE systems:** time-dependent Schrödinger equation;

- **NAS parallel benchmark multi-zone version (NPB-MZ):**
  solvers for discretized versions of the unsteady, compressible Navier-Stokes equations; 256 or 1024 zones of equal size;
advantage of using **orthogonal communication**;
smallest runtimes for **consecutive** and **mixed** (d=4) mapping;
IRK method for sparse ODEs on CHiC – execution time

advantage of using orthogonal communication;
smallest runtimes for consecutive mapping;
IRK method for dense ODEs on JuRoPa – speedup

IRK (s=4) for SCHROED (d=120002) on JuRoPA

data parallel

speedup

computation times are dominating communication times
DIIRK method for dense ODEs on CHiC – execution time

DIIRK (s=4) for SCHROED on CHiC (512 cores)

- data parallel
- task parallel
- task parallel (orthogonal, consecutive mapping)
- task parallel (orthogonal, scattered mapping)
- task parallel (orthogonal, mixed mapping (d=2))

intra M-Task communication more important as for IRK methods
PAB method for sparse ODEs on CHiC – execution time

PAB (K=8) for BRUSS2D on CHiC (256 cores)

- data parallel
- task parallel
- task parallel (orthogonal, consecutive mapping)
- task parallel (orthogonal, scattered mapping)
- task parallel (orthogonal, mixed mapping (d=2))

smallest execution times for mixed mapping
PAB method for sparse ODEs on JuRoPa – execution time

smallest execution times for mixed mapping
PABM method for dense ODEs on CHiC – speedups

largest speedups for consecutive mapping
PABM method for sparse ODEs on JuRoPa – execution time

smallest execution times for consecutive mapping
NAS benchmarks class D on CHiC – GFLOPS/s

1024 zones – best performance for 64 parallel groups with scattered mapping
256 zones – best performance for 128 parallel groups with scattered mapping
NAS benchmarks class C on CHiC – GFLOPS/s

BT–MZ benchmark class C on CHiC (256 cores)

- task parallel (consecutive mapping)
- task parallel (scattered mapping)
- task parallel (mixed mapping, d=2)

256 zones – best performance for 32 or 128 parallel groups with scattered mapping
1024 zones – best performance for 32 or 64 parallel groups with scattered mapping
MPI tasks vs. OpenMP threads

different combinations of MPI processes and OpenMP threads;
8-stage PABM for sparse ODEs on SGI Altix: time per step:

orthogonal: 64 processes and 4 threads minimizes execution time;
Outline

1 Introduction and Motivation

2 Scheduling and Mapping

3 Experimental Evaluation

4 Conclusions
Conclusions

- multi-core systems provide a large number of cores; scalability becomes an important issue for parallel software;
- architecture often has a hierarchical organization; M-task approach for scalability improvement is suitable;
- different strategies for the mapping of M-tasks to physical cores;
- experimental evaluation: significant differences in the performance of different mappings can occur;
  communication requirement of the application and the communication performance of the target platform play an important role;
- orthogonal communication patterns and hybrid MPI+OpenMP implementations can be integrated into the M-tasks approach;
- extension to grid and cloud computing TGrid runtime system;