Message Passing and Threads (Hybrid Codes for HPC)

ComplexHPC Spring School 2013
June 4, 2013

Josef Weidendorfer

Chair for Computer Architecture
Technische Universität München, Munich, Germany
My Background

• Senior researcher at chair for computer architecture (CS faculty, TUM)
  – how to exploit current & future (HPC) systems (multicore, accelerators)
  – programming models, performance analysis tools, application tuning
  – for lecturing & research, use resources of the local compute center LRZ (mostly „standard“ multi-core clusters)

• Interests
  – PhD on load balancing of commercial car crash code (MPI) 2003
  – (cache) analysis tools, application optimization
  – exploitation of (heterogeneous, parallel) architectures
  – virtual machines, JIT-terms, dynamic code generation
Compute Resources at LRZ

• SuperMUC
  – Top500 Rank 4 (Jun12)
  – Linpack: 3,2 PFlop/s
  – 9216 nodes, Infiniband node: 2x Intel E5-2680 (8C, 2.7GHz), 32GB
  – “nice for programmers”
• and smaller other systems
Parallel Programming Models

What is the best model for HPC?

• vendor library
• implicit (we have smart compilers 😊)
• Shared Memory (PThreads, OpenMP, C++11, Cilk, PGAS, CLOMP, DSM, ...)
• Message Passing (PVM, MPI, Go, ...)
• Task-based (OmpSS, StarPU, OpenMP 3/4)
• Accelerator-specific (CUDA, OpenCL, VHDL, ...)

Weidendorfer: Hybrid Programming
Parallel Programming Models

What is the best model for HPC?

Should be
• easy to use
• robust against errors
• composable
Parallel Programming Models

What is the best model for HPC?

No, we want best usage of HW possible
• low level, allowing for controlling details
• compatible with old legacy code
• cover all these new fancy accelerators
• ...
• ok, maybe not too complex
Parallel Programming Models

What is the best model for HPC?

(Still) Status Quo

- MPI between cluster nodes
- MPI on node level if possible
- Accelerator-specific libraries (not needed on SuperMUC)
Future-proof Programming Models

- Energy consumption is main issue
- Power for transfer >> Power for computation
  ➔ avoid data transfers if possible
  ➔ transfers must be controllable at every level

- Message Passing: yes
- Shared Memory: with affinity control, difficult
- Task-based: difficult
Outline

• Why to go Hybrid
  – Advantages, Pitfalls, Examples

• Specifics
  – Hybrid programming structures,
    MPI thread compliance, configuration

• NPB-MZ
Benefits of Shared Memory (1)

• easy access to shared data
  – no domain decomposition needed, concentrate on equal workload splitting
  – fast access to neighbor data on same SMP
  – potential reduction of data transfers (only on demand)
Benefits of Shared Memory (2)

• easy access to shared data
• reduction in resource requirements
  – no duplication of data
  – no explicit consistency enforcement needed
  – use of shared caches for shared data
    • better cache utilization, less pressure on memory
    • allows cache optimizations with multiple cores
      (wavefront among cores with shared caches)
Benefits of Shared Memory (3)

• easy access to shared data
• reduction in resource requirements
• easy load balancing
  – OpenMP self-scheduling schemes (for loops): dynamic, guided
  – OpenMP tasks
• fast synchronization
Benefits of Hybrid

• reduces scalability issues of flat MPI
  – memory requirements of
    • huge communicators
    • buffers for point-to-point communication
  – all-to-all quadratic complexity
• potentially better mapping to hardware
  – if there is a natural hierarchy in app work pieces
Drawbacks of Shared Memory

• synchronization
  – correctness: races possible
  – may have high overhead: lock contention
• beware of (bad) sharing effects
  – false sharing
  – bad cache line utilization
  – sharing among different cores
    (⇒ much coherence traffic with dynamic load balancing)
• locality
  – difficult to control
  – does not fit well with dynamic load balancing
Hybrid MPI/OpenMP

Best mapping of work load to MPI/OpenMP?

e.g. 8 cores: (4/2) or (2/4)?

(\# MPI processes, \# threads per MPI process)
Hybrid MPI/OpenMP

Best mapping of work load to MPI/OpenMP?

- multiple MPI processes per node, per socket?
- hybrid may be slower than pure MPI
  - best choice depends on hardware
  - careful control of locality for optimal exploitation
Example: Raytracing

Memory requirements
• MPI: 3D scene duplicated in each MPI process
• hybrid: 3D scene only once per SMP node

Load balancing
• MPI: difficult (moving scene?)
• hybrid: dynamic load balancing within SMP node
Example: Molecular Dynamics

Particles-in-Cell method
• geometric decomposition
• particles move between cells

MPI
• issues with load balancing
• much time spent in moving particles between MPI processes

Hybrid: easier...
2D Jacobi: Cache Optimizations

- Base version: 1 layer of ghost cells
  - bandwidth requirement: 16 bytes/update
  - memory-bound: 2 cores already occupy bus
2D Jacobi: Cache Optimizations

- Spatial blocking of n iterations: n ghost layers

blocks fit into $\$: update of inner borders

reduced BW to memory $\Rightarrow$ better scalability

- MPI: duplication of ghost layers, redundant comp.
- hybrid: less memory/BW, no redundant comp., enables cache-obliviousness (recursive bisection)
2D Jacobi: Blocking & Redundancy

![Bar charts showing lattice site updates and overhead](image)

- **a)** Lattice site updates of a tile with dimensions $500^2$.
- **b)** Lattice site updates of a block with dimensions $50^3$.

[Wittmann, MA thesis Bamberg, 2009]
2D Jacobi: Cache Optimizations

- Wavefront: similar to blocking, use shared $\$$
  - allows larger blocks, less border updates
  - not possible among MPI processes (matrix needs to be streamed through cores)
OpenMP & ccNUMA

Non Uniform Memory Access

• Memory per socket
  – less latency on local accesses
  – total bandwidth scales with #sockets
OpenMP & ccNUMA

Non Uniform Memory Access

• Memory per socket

• Importance of local accesses
  – needed to exploit full total bandwidth
  – less latency, no contention on inter-chip links
OpenMP & ccNUMA

Non Uniform Memory Access
- Memory per socket
- Importance of local accesses
- Control thread & memory placement
  - threads fixed
  - memory allocated according „first-touch“
  ➔ initialization and data accesses from same thread

Hybrid: Better one MPI process per socket
Pitfalls

• Usage of OpenMP
  – compilers switch off some optimizations to adhere to OpenMP memory model
  – check sequential runtime vs. OpenMP/1 thread
  – icc better than gcc

• Correctness: make sure code is race-free
  – use tools such as Thread Checker / Valgrind

• MPI really thread-safe?

• MPI progress on long calculations using OpenMP?
Outline

• Why to go Hybrid
  – Advantages, Pitfalls, Examples

• Specifics
  – Hybrid programming structures, MPI thread compliance, configuration

• NPB-MZ
Specifics: Thread-safe MPI (1)

„Environment may support threads poorly“

• thread compliance optional in MPI
  – work-around: use global lock?

• check at runtime, different compliance levels
  – MPI_thread_init
  – may switch to (slower?) implementation

• threads in MPI process not addressable

• user must avoid races in MPI
Specifics: Thread-safe MPI (2)

- thread-compliant = thread-safe
- blocking MPI calls only block thread
- MPI_Finalize in same thread as initialization
- threads must not block on same MPI req
- probe/receive
  - „no other matching probe in-between“
  - use locking, or different tags/communicators
- user must ensure correct order of collectives
Specifics: Thread-safe MPI (3)

MPI_Init_thread(int required, int *provided)

Levels of thread-compliance

• MPI_THREAD_SINGLE: not compliant
• MPI_THREAD_FUNNELED: only main thread must call MPI
• MPI_THREAD_SERIALIZED at one point in time, only one thread calls MPI
• MPI_THREAD_MULTIPLE
Switching between MPI / OpenMP phases

- **Initialization of MPI and application**
  - MPI process 1
  - MPI process n

- **OpenMP parallel region**
  - Thr. 1
  - Thr. m

- **Synchronous MPI communication (in OpenMP single/master thread)**
  - Thr. 1
  - Thr. m

- **Finalization of MPI and application**
Specifics: Hybrid Programming (1)

Switching between MPI / OpenMP phases

• MPI communication phase
  – only master/single communicates
    • needs MPI_THREAD_FUNNELED
  – multiple threads per MPI process communicate
    • may result in better network exploitation
    • identify threads by using tags

• issues
  – all other threads sleeping in comm. phase
  – network bandwidth fully exploitable by 1 thread?
Overlap MPI / OpenMP

**Initialization of MPI and application**

**OpenMP parallel region**

- Thr. 1
  - Calculation
  - MPI
- Thr. m
  - Calculation
  - MPI

**Finalization of MPI and application**
Overlap MPI / OpenMP

• split off communication-dependent work
• dedicated thread calls blocking MPI calls
  – makes sure communication progresses
  – difficult with usual OpenMP work sharing
  – better done with OpenMP tasks
• non-blocking MPI calls
  – may have issue with MPI progress
Specifics: Environment

• **NUMA control**
  – fixed threads: control affinity
    • **KMP_AFFINITY** (ICC), **GOMP_CPU_AFFINITY** (GCC)
    • ICC: “scatter” / “compact” / “proclist=[1,2,{3,4},...]”
    • GCC: “1 2 3-4 ...”
    • OpenMP 3.1: **OMP_PROC_BIND=true**
  – explicit memory control & policies
    • “first touch” / explicit control via mbind, numalib
  – may be controlled differently in job scheduling
    • IBM poe: **MP_AFFINITY=“core:$OMP_NUM_THREADS”**
Outline

• Why to go Hybrid
  – Advantages, Pitfalls, Examples

• Specifics
  – Hybrid programming structures, MPI thread compliance, configuration

• NPB-MZ
NAS Parallel Benchmarks

- different scientific „real“ kernels
  - sorting, emb. par., conj. grad., multi-grid, FFT
- three pseudo apps: 3D CFD solvers
  - block tri-diagonal (BT),
    scalar penta-diagonal (SP),
    lower-upper Gauss-Seidel (LU)
  - different workload classes
NPB: Multi-zone Hybrid

Multi-zone hybrid versions (-MZ):
“test the effectiveness of multi-level and hybrid parallelization paradigms and tools”

– BT-MZ (block triagonal)
  • zone sizes vary widely, up to factor 20
  • pure MPI: load balancing issues (if #zones ~ #procs)

– SP-MZ (Scalar Pentadiagonal)
  • zone sizes equal, should scale well
BT-MZ @ SuperMUC

Class A

- **Sequential:** 36.5 s

- **16 threads / 1 node**
  - (1/16) 4.3s
  - (2/8) 3.6s
  - (4/4) **3.4s**
  - (8/2) 6.5s
  - (16/1) 10.5s

- **32 threads / 2 nodes**
  - (2/16) 2.1s
  - (4/8) **1.8s**
  - (8/4) 3.7s
  - (16/2) 7.1s

(MPI/OpenMP)
NPB: Multi-zone Hybrid

Using more OpenMP threads could reduce the memory usage substantially, up to five times on Hopper Cray XT5 (eight-core nodes).

Better integration of MPI & OpenMP

• solution for stalling MPI progress on long OpenMP calculation phases

• specific OpenMP “communication tasks”
  – can be used to check for completion of MPI calls
  – may fail and be re-run at defined intervals
    ➔ regularly calls into MPI
  – if succeeding, enable communication-dependant OpenMP tasks
  – combine with locality-preserving scheduling
References

• MPI Spec, OpenMP Spec
• Other talks on Hybrid Programming
  – SC10/ISC12 tutorials (Rabenseifner, Hager, Jost)
  – Hybrid application studies NERSC
• www.nas.nasa.gov/publications/npb.html

Contact
Josef.Weidendorfer@in.tum.de
Demo / Hands-On

- Demo NBP on SuperMUC
- Measure performance reachable with blocking
- Tradeoff communication volume vs. locality