Runtime analysis of parallel applications for industrial software development

Markus Geimer and Christian Feld, Forschungszentrum Jülich GmbH
Daniel Becker, Siemens AG
Application requirements

- Multicore tool support
- Runtime recording with Score-P
- Supportive tool stack
- Summary
Siemens AG

1) Commonwealth of Independent States

1) Commonwealth of Independent States
Typical multicore migration

Let us assume that an application has already multiple threads...

<table>
<thead>
<tr>
<th>Concurrency bugs</th>
<th>Poor performance</th>
<th>High effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Data races</td>
<td>• Synchronization &amp; communication</td>
<td>• Explicit thread management and synchronization</td>
</tr>
<tr>
<td>• Deadlocks/livelocks</td>
<td>• Contention on shared resources</td>
<td>• Limited portability</td>
</tr>
<tr>
<td>• Nondeterminism</td>
<td>• ...</td>
<td>• ...</td>
</tr>
<tr>
<td>• Memory model</td>
<td>• ...</td>
<td></td>
</tr>
<tr>
<td>• ...</td>
<td>• ...</td>
<td></td>
</tr>
</tbody>
</table>

Multicore migration is extremely challenging
Outline

Application requirements

**Multicore tool support**

- Runtime recording with Score-P
- Supportive tool stack
- Summary
Design process towards multicore software

Tool support is essential for an effective and efficient parallelization
Multicore migration scenario

- Legacy code is code where nobody understands the details
- Runtime behavior is also complex and hard to follow
- Manual extraction of information is cumbersome
- Profiling and tracing tools are essential
  - To understand and debug
  - To engineer and optimize the runtime

Requirements for profiling & tracing systems

- Focus on understanding the application and its parallel aspects
  - Threads & processes
  - Locks & messages
- Portable to Windows, Linux
- HW independent (x86, ARM, and ppc)
- Heterogeneous system support (e.g., Intel Xeon Phi, CUDA)
- Formats enabling interoperability and custom analysis types
Outline

Application requirements
Multicore tool support

Runtime recording with Score-P
Supportive tool stack
Summary
Runtime recording with Score-P

- Open source community
- Linux (& Windows)
- HW independent (x86, ARM, PPC, …)
- Heterogeneous systems (e.g., Intel Xeon Phi, CUDA)
- Open formats enabling interoperability and custom analysis types
- Extremely scalable

www.score-p.org
## Key features

- Provides typical functionality for HPC performance tools
  - Support for process-level parallelism using MPI/SHMEM
  - Support for thread-level parallelism using OpenMP/Pthreads
  - Support for accelerator-level parallelism using CUDA
- Based on instrumentation
  - Supports various techniques
  - Extensive runtime filtering & selective recording capabilities
- Flexible measurement with single re-compilation
  - Basic and advanced profile generation
  - Event trace recording
  - Online access to profiling data
- Scalability: Petascale
- Portability: Supports all major HPC platforms, incl.
  - IBM Blue Gene, Cray XT/XE/XK/XC, Fujitsu FX10 & K computer,
  - SGI Altix, Power/AIX, Linux-based clusters (x86, ARM, Power)
- Open source: 3-clause BSD license

---

**JUQUEEN**

28 rack IBM Blue Gene/Q
28,672 nodes (458,752 cores, 4-way SMT)
448 TB RAM, 5.9 Petaflop/s peak
Score-P workflow

Score-P measurement lib

Instr. target application

HWC

OTF2 event traces

Scalasca Trace Analyzer

CUBE4 summary report

CUBE4 wait-state report

Report manipulation

Optimized measurement configuration

Instrumented executable

Instrumenter compiler / linker

Source modules

Vampir

Cube

TAU ParaProf
# Score-P instrumentation options

<table>
<thead>
<tr>
<th>Manual instrumentation</th>
<th>Automatic instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Extensive API (C/C++/Fortran), supporting</td>
<td>• By using the compiler (GCC, Intel, PGI, IBM xl, Cray, Fujitsu)</td>
</tr>
<tr>
<td>• Program phases</td>
<td>• By using source-to-source translation (Opari2, PDToolkit)</td>
</tr>
<tr>
<td>• Functions</td>
<td>• By linking against a pre-instrumented library (MPI, SHMEM)</td>
</tr>
<tr>
<td>• Arbitrary code regions</td>
<td>• By using GNU linker symbol renaming (POSIX threads, SHMEM)</td>
</tr>
<tr>
<td>• Measurement control</td>
<td></td>
</tr>
</tbody>
</table>
Automatic binary instrumentation using Pin

- Pin: dynamic binary instrumentation tool from Intel
- Flexible instrumentation at image-load time
  - No recompilation necessary
  - But only supports x86 architectures

- **Current prototype supports**
  - Function wrapping
  - Incl. pre-runtime filtering (i.e., file level, region level, shared-object level)
  - Replacement of threading API routines (by calls to Score-P measurement system)
    - POSIX threads
    - Windows threads
    - Qt thread API
    - ACE threads

**Future work: Support for MTAPI (Multicore Association), Intel TBB**
Outline

Application requirements
Multicore tool support
Runtime recording with Score-P

Supportive tool stack

Summary
Cube profile viewer

What kind of performance metric?
Where is it in the source code? In which context?
How is it distributed across the processes/threads?
Open Trace Format 2 (OTF2)

Key facts

- Successor to OTF (Vampir) and EPILOG (Scalasca)
- Very flexible, scalable, and space-efficient multi-file format
- Supports many event types
  - Enter/Exit regions/phases/…
  - Point-to-point, collective, one-sided communication, and synchronization
- Fork/join and create/wait threading
- HW + SW counters
- Extensible event attributes
- Well-defined read/write C API
Time-line visualization in Vampir (TU Dresden)
Automatic trace analysis with Scalasca

- **Idea**
  - Automatic search for patterns of inefficient behavior
  - Classification of behavior & quantification of significance

- **Advantages**
  - Guaranteed to cover the entire event trace
  - Quicker than manual/visual trace analysis
  - Parallel replay analysis exploits available memory & processors to deliver scalability
Example: Lock contention analysis

- Frequent and severe problem during multicore migration
- Automatic determination of waiting times in acquire lock operations
- Easy identification of blocking threads in different call paths

- Currently supports
  - OpenMP critical sections & lock API
  - Pthread mutex & condition variable APIs

- Future work
  - Determine root-cause of waiting time
Outline

Application requirements
Multicore tool support
Runtime recording with Score-P
Supportive tool stack

Summary
Summary

• Parallelizing legacy software is tedious
  • Developers have to understand all parts of the software
  • Getting synchronization right and efficient is challenging

• Profiling and tracing tools
  • Enable developers to understand, debug, engineer, and optimize their application
  • Have to be portable and extendable
  • Should allow to focus only on relevant parts of the application

• Score-P fulfills these requirements and comes with an supportive tool stack enabling effective and efficient multicore migrations
More information and contacts

Multicore Expert Center, Siemens AG
  • Dr. Daniel Becker
  • becker.daniel@siemens.com

Score-P & OTF2
  • http://www.score-p.org
  • support@scorep.org

Scalasca & Cube
  • http://www.scalasca.org
  • scalasca@fz-juelich.de