Review

Cédric Valensi
University of Versailles

Summary

You've been introduced to a variety of tools

- with hints to apply and use the tools effectively

Tools provide complementary capabilities

- computational kernel & processor analyses
- communication/synchronization analyses
- load-balance, scheduling, scaling, …

Tools are designed with various trade-offs

- general-purpose versus specialized
- platform-specific versus agnostic
- simple/basic versus complex/powerful

Tool selection

Which tools you use and when you use them likely to depend on the situation

- which are available on (or for) your computer system
- which support your programming paradigms and languages
- which you are familiar (comfortable) with using
- which type of issue you suspect
- which question you want to have answered

Being aware of (potentially) available tools and their capabilities can help finding the most appropriate tools

Workflow (getting started)

First ensure that the parallel application runs correctly

- no-one will care how quickly you can get invalid answers or produce a set of core-files
- parallel debuggers like DDT or STAT help isolate anomalies
- correctness checking tools such as *MUST/ARCHER* can identify other issues (that might not cause problems right now, but will eventually)
  - e.g., race conditions, invalid/non-compliant usage

Best to start with an overview of execution performance

- fraction of time spent in computation vs. communication/synchronization vs. I/O
- which sections of the application/library code are most costly
- Example parallel profilers: **Score-P + Cube/ParaProf, TAU**, **MAQAO**, *MAP*

and how it changes with scale or different configurations

- processes vs threads, mappings, bindings

Workflow (communication/synchronization)

Communication issues generally apply to every computer system (to different extents) and typically grow with the number of processes/threads

- *Weak scaling*: fixed computation per thread, and perhaps fixed localities, but increasingly distributed
- *Strong scaling*: constant total computation, increasingly divided amongst threads, while communication grows
- Collective communication (particularly of type “all-to-all”) result in increasing data movement
- Synchronizations of larger groups are increasingly costly
- Load-balancing becomes increasingly challenging, and imbalances more expensive
  - generally manifests as waiting time at subsequent collective operations

Workflow (wasted waiting time)

Waiting times are difficult to determine in basic profiles

- Part of the time each process/thread spends in communication & synchronization operations may be wasted waiting time
- Need to correlate event times between processes/threads
  - *Periscope* uses augmented messages to transfer timestamps plus on-line analysis processes
  - Post-mortem event trace analysis avoids interference and provides a complete history
  - **Scalasca** automates trace analysis and ensures waiting times are completely quantified
  - **Vampir** allows interactive exploration and detailed examination of reasons for inefficiencies
  - *BSCtools* (Paraver/Dimemas/Extrae) support comprehensive detailed trace analyses

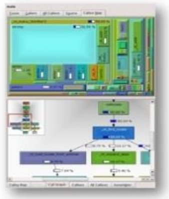
Workflow (core computation)

Effective computation within processors/cores is also vital

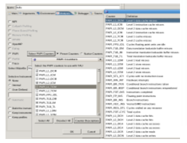
- Optimized libraries may already be available
- Optimizing compilers can also do a lot
  - provided the code is clearly written and not too complex
  - appropriate directives and other hints can also help
- Processor hardware counters can also provide insight
  - although hardware-specific interpretation required
  - can be accessed with **PAPI** or **LIKWID**
- Cache analysis can be investigated with *kcachegrind*
- Optimised code generation for x86 can be investigated with **MAQAO**
- Tools available from processor and system vendors help navigate and interpret processor-specific performance issues

Technologies and their integration

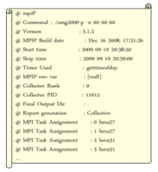
KCACHEGRIND



PAPI

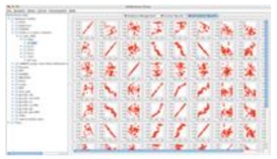


MAP/PR / MPIP / O|SS / **MAQAO** / **LIKWID**

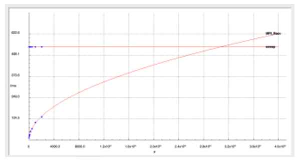


```
@ mpiP
@ Command : ./smg2000-p -n 60 60 60
@ Version                 : 3.1.2
@ MPIP Build date          : Dec 16 2008, 17:31:26
@ Start time              : 2009 09 19 20:38:50
@ Stop time               : 2009 09 19 20:39:00
@ Timer Used               : gettimeofday
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 11612
@ Final Output Dir         : .
@ Report generation        : Collective
@ MPI Task Assignment       : 0 hera27
@ MPI Task Assignment       : 1 hera27
@ MPI Task Assignment       : 2 hera31
@ MPI Task Assignment       : 3 hera31
...
```

TAU

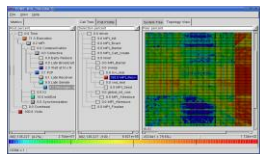


EXTRA-P

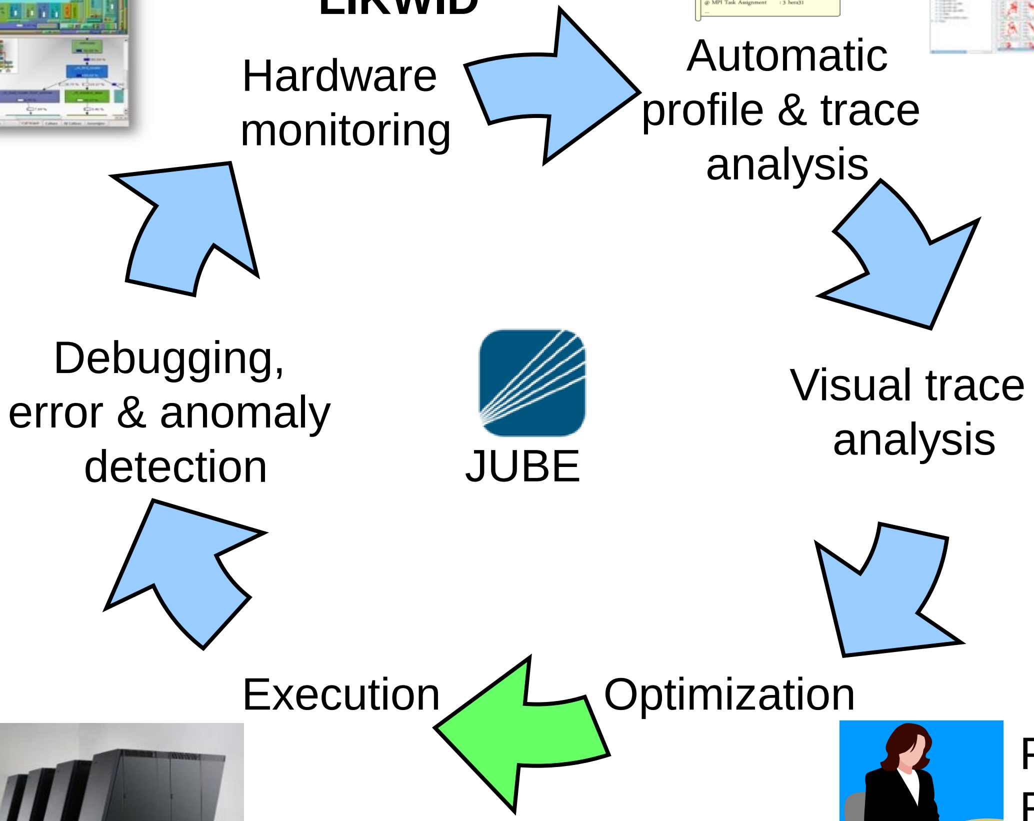


PERISCOPE

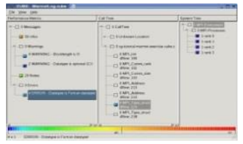
SCALASCA



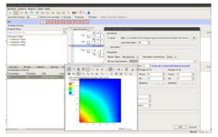
SCORE-P / EXTRAE


Hardware monitoring
Automatic profile & trace analysis
Visual trace analysis
Optimization
Execution
Debugging, error & anomaly detection
JUBE

MUST / ARCHER

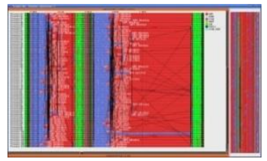


DDT

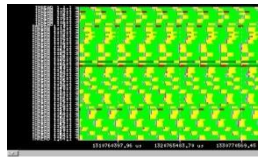


STAT

VAMPIR



PARAVER



MEMCHECKER / SPINDLE / SIONLIB

PTF / RUBIK / **MAQAO**

Evaluation / Feedback

- Please also complete the online VI-HPS workshop/tools evaluation form
  - provides valuable feedback
    - to improve future workshops and training
    - to tools developers for improving their tools and training material
  - can be anonymous if desired
  - **https://forms.gle/EcMimEUZZDh5uJtw8**
- Tools support queries and bug reports are also welcome
  - should be submitted to respective support mailing lists
  - and/or sometimes to local technical support staff

Further information

Website

- Introductory information about the VI-HPS portfolio of tools for high-productivity parallel application development
  - VI-HPS Tools Guide
  - links to individual tools sites for details and download
- Training material
  - tutorial slides
  - latest ISO image of VI-HPS Linux DVD with productivity tools
  - user guides and reference manuals for tools
- News of upcoming events
  - tutorials and workshops
  - mailing-list sign-up for announcements

http://www.vi-hps.org