Parallel Performance Analysis
Using the Paraver Toolkit

Rainer Keller
University of Stuttgart
High-Performance Computing Center Stuttgart (HLRS)
http://www.hlrs.de

Overview

- Motivation
- Overview Performance Measurement
- Getting started with Paraver & mpitrace
- Paraver Windows and Modules
- Paraver analysing CPMD on SX-8
- Paraver’s real power on Hardware-Performance Counter Analysis
- Summary
Motivation

• Many ways to represent the calculation of a physical phenomenon
• It is hard to make the right coding decisions at the design phase
• Harder to optimize for a specific architecture
• Even harder to keep the code optimal over generations of developers and hardware architectures
• Now consider parallel execution and architectures

Efficiency

• There are many ways to describe efficiency of applications
  – Numerical efficiency
  – Computational overhead
  – Parallel efficiency
• With regard to parallel efficiency, consider
  – Amdahl’s law
  – ...

Reality
Physical Model
Mathematical Model
Numerical Scheme
Algorithmic Description
A few parallel Programming Models e.g. MPI, OpenMP
Hardware Architecture
Overview Performance Measurement 1/5

- System level performance:
  - CPU: top
  - Memory / IO: vmstat
  - Network: ifconfig or ibstat

Overview Performance Measurement 2/5

We need more detailed information for Optimization.

We need better overview through visualization & filtering of information.

Especially holds true for

- Parallel Programming Models e.g. MPI, OpenMP
- Hardware Architecture

- For Single-Processor: gprof, vtune, speedshop, cachegrind, PAPI
- For Multi-Processor: Tau, Vampir, Paraver
Overview Performance Measurement 3/5

- Code instrumentation (Manual):
  - Manually enclose interesting regions:

  **Standard way (Posix 1003.1):**
  ```c
  #include <sys/time.h>
  int64_t gettimenow (void) {
    struct timeval now;
    gettimeofday (&now, NULL);
    return now.tv_sec*1000*1000+
        now.tv_usec;
  }
  ```

  ```c
  double start, end;
  start = gettimenow();
  do_some_heavy_calc(…);
  end = gettimenow();
  ```

  **Standard way using MPI:**
  ```c
  #include "mpi.h"
  ```

  ```c
  double start, end;
  start = MPI_Wtime();
  do_some_heavy_calc(…);
  end = MPI_Wtime();
  ```

  - Low overhead, one knows, when, what is measured.
  - Very time-consuming to instrument, need tools to analyse.

Overview Performance Measurement 4/5

- Code instrumentation (Automatic) at Compile-step:
  - Preprocessor adding function calls at function entry&exit.
  - Compiler generating function calls at function entry&exit:
    - **icc:** `-prof-gen` [x]
    - **gcc:** `-p` / `-pg` or `-finstrument-functions` and write timing funcs.

  ```c
  double do (int i) {
    _cyg_profile_func_enter(&do,0x...);
    ...
    _cyg_profile_func_exit(&do,0x...);
  }
  ```

  ```c
  double iter (int n) {
    _cyg_profile_func_enter(&iter,0x...);
    while (--n > 0) {
      err = do (n);
      if (err < EPS) break;
    } _cyg_profile_func_exit(&iter,0x...);
  }
  ```

  - Low overhead (depends on timing routines), very powerful method
Overview Performance Measurement

- Binary Code instrumentation (Automatic) at Link/Execution-step:
  - Link with Wrapper libraries or
  - For dynamic libraries use a dynamic library with wrappers:
    export LD_PRELOAD=libmyfunctions.so
    export LD_LIBRARY_PATH=/opt/timing_libraries
- For MPI: PMPI-Interface:
  Every MPI Function is callable through equivalent PMPI-name.

```c
int MPI_Send (void* buf, int count, MPI_Datatype type,
              int dest, int tag, MPI_Comm comm) {
    double start, stop;
    int ret;
    start = MPI_Wtime ();
    ret = PMPI_Send (buf, count, type, dest, tag, comm);
    stop = MPI_Wtime ();
    store_information (MPI_SEND, my_rank, dest, tag,
                       gettype_size(type) * count, getcomm_id(comm));
    return ret;
}
```

Getting started using Paraver & libmpitrace

- Performance of parallel processes may be analysed post-mortem using Paraver on generated tracefile (MPI, OpenMP, AIXtrace).
- To generate tracefile, relink the application:
  gcc -o app_trace *.o -lmpitrace -lpmpi -lmpi
  or
  g77 -o app_trace *.o -lmpitracef -lpmpi -lmpi
- Set the environment with enabled tracing:
  export MPITRACE_ON=1
  export MPITRACE_PROGRAM_NAME="solala" /* Optional */
  export MPTRACK_COUNTERS=0x80000034,... /* Optional */
- Run with the tracing-functionality enabled:
  mpirun -np 2 ./app_trace
  mpirace: Tracing enabled for process 5711 (counters enabled).
  mpirace: Tracing enabled for process 5713 (counters enabled).
- Afterwards merge mpit-files generated by each process:
  mpi2prv -o app_trace.prv TRACE*.mpitr
  generates Paraver-Tracefile (prv) and a description file (pcf).
Getting started using Paraver & libmpitrace  2/2

- The Paraver-Tracefile (prv) contains plain-text timestamps with states, event-type, rank, dest information:

```
#Paraver (28/06/06 at 2:34):41660812:8:1:8(1:1,1:2,1:3,1:4,1:5,1:6,1:7,1:8),2
c:1:92:1:-1
2:1:1:1:1:0:40000001:1
1:1:1:1:1:0:20:1
1:2:1:2:1:0:8748:2
```

- The Paraver Description File maps the states&events to names (used later for clear-text and semantic analysis):

```
EVENT_TYPE
9    50000002    MPI Collective Comm
VALUES
10   MPI_Allreduce
8    MPI_Barrier
7    MPI_Bcast
0    End
```

Paraver: First Startup

Main Window
- Timing Window
- Visualizer Module:
  - Trace Selection
  - Function Handling
- Filter Module:
  - Select Values
  - Tag, Rank
  - HW Perf-Counter
- Global Controller:
  - To open Modules
- Semantic Module:
  - How function is interpreted

Always remember: Everything is just a function of Time.
Paraver: Windows and Modules

- Open Semantic Module
- Open Visualizer Module
- Open Filter Module
- Play Forward/Backward
- Open 2D Analysis Module

Trace-file handling:
- Load new (multiple) traces
- Unload currently not recommended (just close win).

Trace information:
- On the number of processes
- Processes / node
- Sates, Events and so on

Trace Generation:
- Safe section for Paraver & Dimemas
Paraver: Windows and Modules

- A trace is a function over time with semantic interpretation:

  ![Trace Diagram]

  - Eight Processes (each w/ 1 thread)
  - Flags indicate Events (green)
  - P2P comm (yellow)
  - (Un)Display Communication
  - Display numeric Value of function
  - Slider to go through time

Paraver: States

- For MPI, the following states are color-defined (selection):
  - (blue) Running
  - (red) Waiting for a message
  - (pink) Blocking send
  - (pink) Immediate send
  - (gray) Immediate receive
  - (orange) Collective communication
  - (brown) Synchronization / Barrier

- For color-gradients (and min and max):
Paraver: CPMD on SX8 – The big Picture

- Analyse the well-known Car-Parrinello Molecular Dynamics code.
- This code runs very well on NEC SX8 due to DGEMM.
- First zoom into the application to figure out communication (we have seen already the massive collective communication – which in the Display overlays the computation)

Communication to rank zero, followed by Barrier.

Zooming into P2P:
This communication is done in linear fashion...

Paraver: CPMD on SX8 – Main computational part

- Zoom into main section: the computational part still is high:
  (95 ms, followed by several MPI_Alltoall and MPI_Allreduce)
• See the (absolute) number of say Branch Mispredictions:

1.) Clone window
2.) Filter for Event
   Select(=) and Number
3.) Choose the semantic
   Interpretation, here:
   Next Event Value
4.) Select the Gradient
   color type (w/ right
   mouse button)
5.) Recalculate the Min &
   Max Y-Value (F)
6.) Apply the whole thing
   (may be partially done
   by pressing OKs)

• Well, that was mighty interesting, but not very sensible…

• Relative values: Semantic Interpretation “Avg Next Event Value”

• One thing very important for a Vector machine is „-]?

Vectorization

• Use Vector Element Counter (VE) & 2D-Analyzer for distribution:

10 to 19% of the vector instructions
have a Vector Length
0 to 16 elements.

26 to 29% of the vector instructions
have a Vector Length
250 to 256 elements.
Paraver: CPMD on SX8 – HW Performance counter 3/3

- Power of Paraver: Filtering for parts of the trace within boundaries of values, e.g. parts, where number of branch mispredictions are within 30 and 75%:

Paraver: CPMD on SX8 – Configurations

- Analysis / Windows may be stored & loaded in Configurations:

For SX of interest:
- Avg. Vector Length
- Bank Conflicts/s
- MFLops (all Units)
- MIPS
- Clocks/s wasted due to Cache-Misses
Paraver: Summary

- Paraver is a versatile tool for Performance Measurement
- However, it’s not very easy to learn – but straightforward…
- There’s a bunch more features to discover…
- Nevertheless, I hope to have raised Your interest in the tool

Thank You very much for Your attention.