PerfExpert :: deep dive

Ashay Rane

Martin Burtscher

James Browne

This work is funded in part by TACC and NSF under OCI award #0622780

Topics in this session

- Steps to use PerfExpert
- Demonstration using an example

Performance optimization using PerfExpert

1. Choosing what to measure
2. Measuring system performance
3. Selecting the application's input data size
4. Measuring application performance
5. Diagnosis of performance problems
6. Suggestions for optimization

Performance optimization using PerfExpert

1. Choosing what to measure
2. Measuring system performance
3. Selecting the application's input data size
4. Measuring application performance
5. Diagnosis of performance problems
6. Suggestions for optimization

#1: Choosing what to measure

- 350+ performance events on most consumer laptops, more on servers
- Often times hard to decipher
- May differ across machines

PerfExpert mines the available events and chooses relevant ones

#2: Measuring system performance

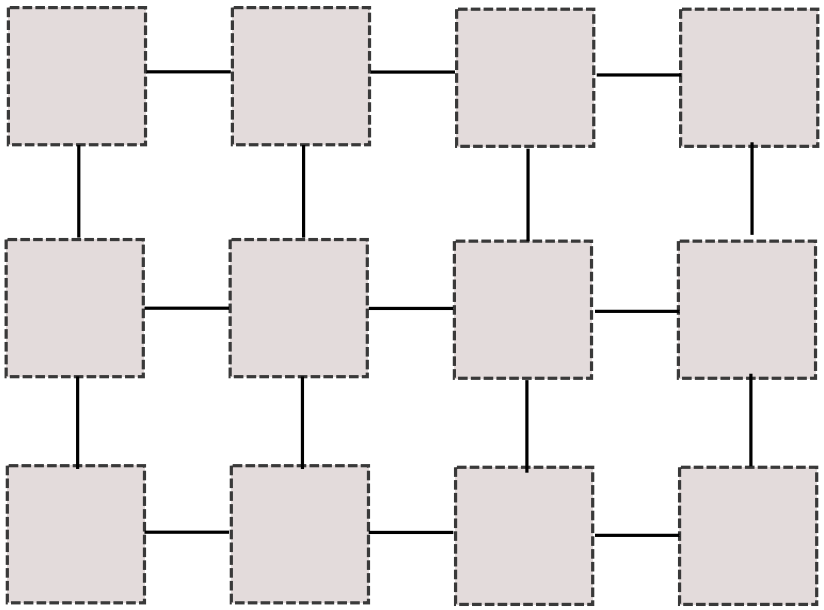
PerfExpert runs micro-benchmarks to measure:

- Cahe latencies
- CPU speed
- Best-case instruction execution times

Results are used to analyze application performance on the specific hardware

#3: Selecting application data size

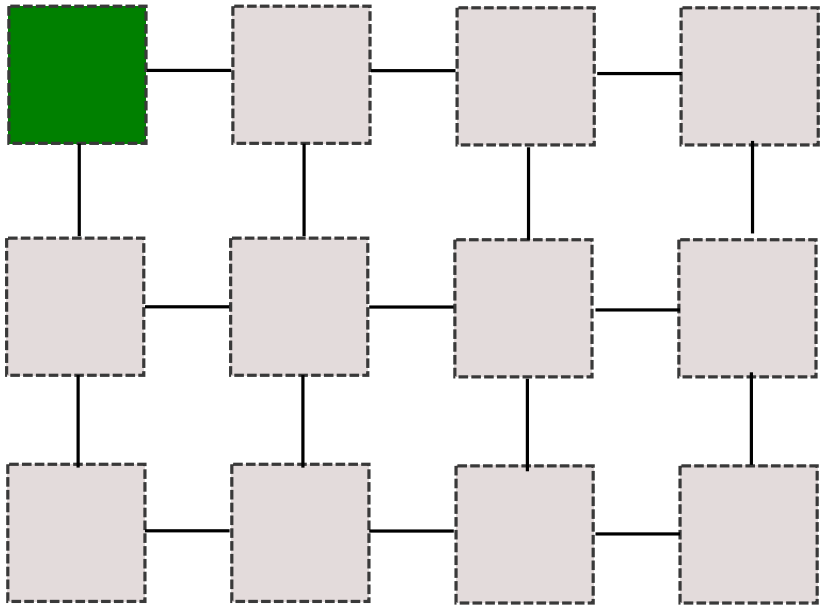
- PerfExpert measures intra-node performance, hence multi-node execution does not provide useful information for profiling



MPI tasks

#3: Selecting application data size

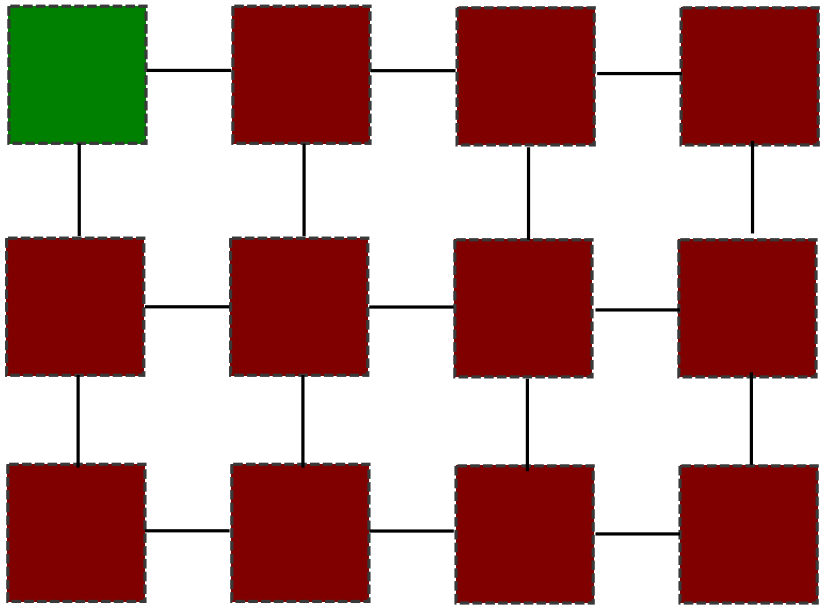
- PerfExpert measures intra-node performance, hence multi-node execution does not provide useful information for profiling



Intra-node performance profiling

#3: Selecting application data size

- PerfExpert measures intra-node performance, hence multi-node execution does not provide useful information for profiling



Intra-node performance profiling

#3: Selecting application data size

- For most applications, possible to use a single-node problem size that is representative of production scale
- May reduce unnecessary overhead

#4: Measuring application performance

- PerfExpert internally uses HPCToolkit. Hence all of the information that HPCToolkit measures is available to PerfExpert but only limited portion of it is used

- PerfExpert runs application multiple (5-6) times (because of limited hardware counters)

- Summarizes measurements in experiment.xml

#5: Diagnosis of performance problems

- PerfExpert derives commonly-used analysis metrics
- Combines performance information from HPCToolkit and system latencies from PerfExpert microbenchmarks
- Six categories of performance breakdown (explained in next slide)
- Metrics are scaled so that they range from good to okay to bad to worse

Performance categories

(What do these categories indicate?)

- **Data access**: Data reuse
- **Data TLB**: Memory access patterns
- **Instruction access, TLB**: Code locality
- **Branch execution**: '`if`' statements
- **Floating-point computation**: Arithmetic operations

http://www.tacc.utexas.edu/perfexpert/metrics/

PerfExpert output

```
Loop in function compute() (99.9% of the total runtime)
================================================================================
ratio to total instrns       %  0.......25.........50.......75......100
  - floating point      :    6 ***
  - data accesses       :   33 ******************
 GFLOPS (% max)         :    7 ***
--------------------------------------------------------------------------------
performance assessment     LCPI good....okay....fair....poor....bad...
* overall                :  0.8 >>>>>>>>>>>>>>>
upper bound estimates
* data accesses          :  2.4 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - L1d hits            :  1.3 >>>>>>>>>>>>>>>>>>>>>>>>>
   - L2d hits            :  0.3 >>>>>
   - L2d misses          :  0.8 >>>>>>>>>>>>>>>>
* instruction accesses   :  0.3 >>>>>>
   - L1i hits            :  0.3 >>>>>>
   - L2i hits            :  0.0 >
   - L2i misses          :  0.0 >
* data TLB               :  1.5 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
* instruction TLB        :  0.0 >
* branch instructions    :  0.0 >
   - correctly predicted :  0.0 >
   - mispredicted        :  0.0 >
* floating-point instr   :  0.2 >>>>
   - fast FP instr       :  0.2 >>>>
   - slow FP instr       :  0.0 >
```

#6: Suggestions for optimization

- PerfExpert contains a database of common problems, their resulting performance metrics and solutions
- Pattern matches performance metrics of user code to those in the database
- Database is growing, not an exhaustive list yet

Demonstration

(using Matrix multiplication code on Ranger as example)

```
void compute()
{
   register int i, j, k;
   for (i = 0; i < n; i++)
      for (j = 0; j < n; j++)
         for (k = 0; k < n; k++)
            c[i][j] += a[i][k] * b[k][j];
}
```

Discovering relevant performance events

- Automated during installation of PerfExpert
- For example, events chosen on Ranger:

```
PAPI_TOT_CYC
PAPI_L1_DCA
PAPI_L2_DCA
PAPI_TOT_INS
PAPI_L1_ICA
PAPI_L2_ICA
PAPI_L2_DCM
PAPI_L2_ICM
PAPI_TLB_DM
PAPI_TLB_IM
PAPI_BR_INS
PAPI_BR_MSP
PAPI_FML_INS
PAPI_FDV_INS
PAPI_FAD_INS
RETIRED_SSE_OPERATIONS:ALL
PAPI_L3_TCM
```

Measuring system performance

- Automated during installation of PerfExpert

- For example, system latencies on Ranger:

```
L1_dlat = 1.02
L1_ilat = 1.02
L2_lat = 18.50
L3_lat = 27.39
mem_lat = 291.59
CPU_freq = 2300003000
FP_lat = 3.00
FP_slow_lat = 28.50
BR_lat = 3.00
BR_miss_lat = 22.53
TLB_lat = 49.52
```

Application performance measurement

- Load appropriate modules:
  `module load papi java perfexpert`

- Run measurements:
  `perfexpert_run_exp ./application.exe`

- Produces XML file containing measurements

Diagnosis

```
perfexpert <threshold> experiment.xml
```

```
Loop in function compute() (99.9% of the total runtime)
================================================================================
ratio to total instrns       %  0.......25.........50.......75......100
  - floating point      :    6 ***
  - data accesses       :   33 ******************
 GFLOPS (% max)         :    7 ***
--------------------------------------------------------------------------------
performance assessment     LCPI good....okay....fair....poor....bad...
* overall                :  0.8 >>>>>>>>>>>>>>>
upper bound estimates
* data accesses          :  2.4 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - L1d hits            :  1.3 >>>>>>>>>>>>>>>>>>>>>>>>>
   - L2d hits            :  0.3 >>>>>
   - L2d misses          :  0.8 >>>>>>>>>>>>>>>>
* instruction accesses   :  0.3 >>>>>>
   - L1i hits            :  0.3 >>>>>>
   - L2i hits            :  0.0 >
   - L2i misses          :  0.0 >
* data TLB               :  1.5 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
* instruction TLB        :  0.0 >
* branch instructions    :  0.0 >
   - correctly predicted :  0.0 >
   - mispredicted        :  0.0 >
* floating-point instr   :  0.2 >>>>
   - fast FP instr       :  0.2 >>>>
   - slow FP instr       :  0.0 >
```

Suggestion by PerfExpert

```
perfexpert --recommend <threshold> experiment.xml
```

```
Loop in function compute() (100% of the total runtime)
=======================================================
Change the order of loops
This optimization may improve the memory access pattern
and make it more cache and TLB friendly.
```

New code

```
void compute()
{
   register int i, j, k;
   for (i = 0; i < n; i++)
      for (k = 0; k < n; k++)
         for (j = 0; j < n; j++)
            c[i][j] += a[i][k] * b[k][j];
}
```

Revised code's performance

```
perfexpert <threshold> experiment-opt.xml
```

```
Loop in function compute() (99.9% of the total runtime)
================================================================================
ratio to total instrns         % 0.......25.........50.......75......100
  - floating point      :      6 ***
  - data accesses       :     33 ******************
 GFLOPS (% max)         :      7 ***
--------------------------------------------------------------------------------
performance assessment     LCPI good....okay....fair....poor....bad....
* overall                :  0.6 >>>>>>>
upper bound estimates
* data accesses          :  0.6 >>>>>>>>>>>>>
   - L1d hits            :  0.6 >>>>>>>>>>>
   - L2d hits            :  0.1 >>
   - L2d misses          :  0.0 >
   - L3d misses          :  0.0 >
* instruction accesses   :  0.2 >>>>
   - L1i hits            :  0.2 >>>>
   - L2i hits            :  0.0 >
   - L2i misses          :  0.0 >
* data TLB               :  0.0 >
* instruction TLB        :  0.0 >
* branch instructions    :  0.2 >>>>>
   - correctly predicted :  0.2 >>>>>
   - mispredicted        :  0.0 >
* floating-point instr   :  0.2 >>>>>
   - fast FP instr       :  0.2 >>>>>
   - slow FP instr       :  0.0 >
```

Commands to use PerfExpert

1. `module load papi java perfexpert`
2. `perfexpert_run_exp ./application.exe`
3. `perfexpert <threshold-between-0-and-1> experiment.xml`
4. `perfexpert --recommend <threshold> experiment.xml`