OpenMP Performance on Virtual Machines

Jie Tao
Karlsruhe Institute of Technology
Outline

- Introduction
  - Virtualization
  - OpenMP
- Test Environment
- Initial Performance
- Performance Analysis Using ompP
- Optimization Results
- Conclusions
Virtualization

- Running multiple OSs on the same hardware

- Basic terms
  - Hypervisor (xen, KVM, VMware)
  - Full vs. Para virtualization

- Adopted for
  - Server consolidation
  - Cloud Computing: on-demand resource provision

- Performance loss
The OpenMP Programming Model

- Programming interface for multiprocessor systems with a shared memory
- Developed by OpenMP Forum
- Standardized, portable
- Supports Fortran, C and C++
- API is based on directives, runtime routine and environment variables
  - PARALLEL (for), SECTION, SINGLE, REDUCTION, BARRIER, LOCK/UNLOCK, ....
  - omp_set_num_threads, omp_get_num_procs, ...
  - OMP_NUM_THREADS, OMP_SCHEDULE, ...
The OpenMP Programming Model (cont.)

Example

Program

```c
#include <stdio.h>
#define LAST 1000
int main()
{
    int i, sum = 0;
    #pragma omp parallel for reduction(+:sum)
    for ( i = 1; i <= LAST; i++ )
    {
        sum += i;
    }
    printf("sum = %d\n", sum);
}
```

Compiling:

```
gcc -fopenmp -o example example.c
```

Execution

```
$ export OMP_NUM_THREADS=2
$ ./example
```

Diagram:

```
master thread

Parallel {  
  FORK  
  sum  
  sum  
  sum  
  barrier
  }  

JOIN

master thread
```

Jie Tao

Vorlesung Parallele Architekturen und Programmierung - WS 06/07
Experimental Setup

- **Host machine**
  - AMD multicore
  - Opteron\(^{\text{tm}}\) Processor 2376
  - Scientific Linux
  - Virtualized with xen and KVM

- **Virtual machines**
  - Hypervisor: xen
  - OS: Debian 2.6.26
  - Compiler: gcc 4.3.2
  - #cores: 1-8
  - Memory: 4GB

- **Benchmarks**
  - SPEC OpenMP
  - NAS OpenMP
  - OpenMP Microbenchmarks
Speedup

**EP**

**FT**

**wupwise**

**equake**
Execution Time - SPEC

**app LU**
- **Host**
- **VM-full**
- **VM-para**

**equake**
- **Host**
- **VM-full**
- **VM-para**

**wupwise**
- **Host**
- **VM-full**
- **VM-para**

**swim**
- **Host**
- **VM-full**
- **VM-para**
Execution Time - NAS

- **BT**
  - Host
  - VM-full
  - VM-par

- **CG**
  - Host
  - VM-full
  - VM-par

- **FT**
  - Host
  - VM-full
  - VM-par

- **EP**
  - Host
  - VM-full
  - VM-par

(Number of cores: 1, 2, 4, 6, 8)
Execution Time – NAS/SP

![Chart showing execution time in seconds for different numbers of cores]

- **SP**
  - Host
  - VM-full
  - VM-para

Execution time in seconds

Number of cores
Performance Analysis with ompP

- ompP
  - A profiling tool based on source instrumentation
  - Delivers overhead analysis reports
  - Also supports performance measurement of hardware counters

- Overheads categories
  - **Synchronization**: Overheads that arise because threads need to coordinate their activity, e.g. critical section or lock
  - **Load imbalance**: Overhead due to different amounts of work performed by threads, e.g. in work-sharing regions
  - **Limited parallelism**: Overhead resulting from unparallelized or only partly parallelized regions of code
  - **Thread management**: Time spent by the runtime system for managing the application’s threads.
## Runtime Overhead of NAS Applications

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Overhead (%)</th>
<th>Synch</th>
<th>Imbal</th>
<th>Limpar</th>
<th>Mgmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT-host</td>
<td>1253.71</td>
<td>81.23 (6.48)</td>
<td>0.00</td>
<td>80.87</td>
<td>0.00</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>1294.55</td>
<td>148.48 (11.47)</td>
<td>0.00</td>
<td>148.47</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>1400.50</td>
<td>163.66 (11.65)</td>
<td>0.00</td>
<td>163.64</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td>FT-host</td>
<td>72.27</td>
<td>25.62 (35.44)</td>
<td>0.01</td>
<td>1.06</td>
<td>24.43</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td>75.02</td>
<td>25.97 (34.53)</td>
<td>0.01</td>
<td>1.04</td>
<td>24.85</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>88.67</td>
<td>32.22 (36.34)</td>
<td>0.00</td>
<td>6.45</td>
<td>25.73</td>
<td>0.04</td>
</tr>
<tr>
<td>CG-host</td>
<td>14.36</td>
<td>1.55 (8.95)</td>
<td>0.00</td>
<td>0.95</td>
<td>0.19</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>17.64</td>
<td>4.87 (23.59)</td>
<td>0.00</td>
<td>3.46</td>
<td>1.37</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>24.05</td>
<td>6.37 (26.49)</td>
<td>0.00</td>
<td>5.27</td>
<td>1.08</td>
<td>0.02</td>
</tr>
<tr>
<td>EP-host</td>
<td>92.27</td>
<td>1.08 (1.17)</td>
<td>0.00</td>
<td>0.93</td>
<td>0.00</td>
<td>0.15</td>
</tr>
<tr>
<td></td>
<td>89.66</td>
<td>1.24 (1.37)</td>
<td>0.00</td>
<td>0.75</td>
<td>0.00</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>133.76</td>
<td>29.60 (22.13)</td>
<td>0.00</td>
<td>29.32</td>
<td>0.00</td>
<td>0.27</td>
</tr>
<tr>
<td>SP-host</td>
<td>4994.76</td>
<td>1652.66 (33.03)</td>
<td>0.11</td>
<td>1651.95</td>
<td>0.00</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>16466.47</td>
<td>14315.84 (86.89)</td>
<td>1.45</td>
<td>14314.36</td>
<td>0.00</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td>6816.17</td>
<td>5302.04 (77.68)</td>
<td>2.74</td>
<td>5299.29</td>
<td>0.00</td>
<td>0.01</td>
</tr>
</tbody>
</table>
### Region Overview of NAS Applications

<table>
<thead>
<tr>
<th></th>
<th>PARALLEL</th>
<th>LOOP</th>
<th>SINGLE</th>
<th>BARRIER</th>
<th>CRITICAL</th>
<th>MASTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>2</td>
<td>54</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>FT</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CG</td>
<td>2</td>
<td>22</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>EP</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SP</td>
<td>2</td>
<td>69</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

**ompP report of a LOOP in sp.c (line 898-906) on the para-virtualized machine**

<table>
<thead>
<tr>
<th>TID</th>
<th>execT</th>
<th>execC</th>
<th>bodyT</th>
<th>exitBarT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>310.60</td>
<td>1541444</td>
<td>11.24</td>
<td>289.41</td>
</tr>
<tr>
<td>1</td>
<td>310.50</td>
<td>1541444</td>
<td>11.22</td>
<td>289.35</td>
</tr>
<tr>
<td>2</td>
<td>310.44</td>
<td>1541444</td>
<td>11.33</td>
<td>289.12</td>
</tr>
<tr>
<td>3</td>
<td>310.26</td>
<td>1541444</td>
<td>11.22</td>
<td>289.14</td>
</tr>
<tr>
<td>4</td>
<td>310.85</td>
<td>1541444</td>
<td>11.26</td>
<td>289.68</td>
</tr>
<tr>
<td>5</td>
<td>310.82</td>
<td>1541444</td>
<td>11.24</td>
<td>289.62</td>
</tr>
<tr>
<td>6</td>
<td>311.10</td>
<td>1541444</td>
<td>11.17</td>
<td>289.99</td>
</tr>
<tr>
<td>7</td>
<td>311.14</td>
<td>1541444</td>
<td>10.92</td>
<td>290.48</td>
</tr>
<tr>
<td>SUM</td>
<td>2485.71</td>
<td>12331552</td>
<td>89.60</td>
<td>2316.76</td>
</tr>
</tbody>
</table>
for (j = 1; j <= grid_points[1]-2; j++) {
    for (k = 1; k <= grid_points[2]-2; k++) {
        #pragma omp for
        for (i = 0; i <= grid_points[0]-1; i++) {
            ru1 = c3c4*rho_i[i][j][k];
            cv[i] = us[i][j][k];
            rhon[i] = max(dx2+con43*ru1,
                          max(dx5+c1c5*ru1,
                              max(dxmax+ru1,
                                  dx1)));
        }
    }
}

#pragma omp for
for (i = 1; i <= grid_points[0]-2; i++) {
    for (j = 0; j <= grid_points[1]-1; j++) {
        for (k = 0; k <= grid_points[2]-1; k++) {
            #pragma omp for
            for (i = 0; i <= grid_points[0]-1; i++) {
                ru1 = c3c4*rho_i[i][j][k];
                cv[i] = us[i][j][k];
                rhon[i] = max(dx2+con43*ru1,
                              max(dx5+c1c5*ru1,
                                  max(dxmax+ru1,
                                      dx1)));
            }
            for (i = 1; i <= grid_points[0]-2; i++) {
                lhs[0][i][j][k] =   0.0;
                lhs[1][i][j][k] = - dttx2 * cv[i-1] -
                                  dttx1 * rhon[i-1];
                lhs[2][i][j][k] =   1.0 + c2dttx1 *
                                    rhon[i];
                lhs[3][i][j][k] =   dttx2 * cv[i+1] - dttx1 *
                                    rhon[i+1];
                lhs[4][i][j][k] =   0.0;
            }
        }
    }
}
Optimization Results

**SP**
- Execution time in seconds
- Number of cores: 1, 2, 4, 6, 8
- Graph showing execution time for Host, VM-full, and VM-para

**SP**
- Speedup
- Number of cores: 1, 2, 4, 6, 8
- Graph showing speedup for Host, VM-full, and VM-para

**SP-OPT**
- Execution time in seconds
- Number of cores: 1, 2, 4, 6, 8
- Graph showing execution time for Host, VM-full, and VM-para

**SP-OPT**
- Speedup
- Number of cores: 1, 2, 4, 6, 8
- Graph showing speedup for Host, VM-full, and VM-para
Conclusions

- Virtualization introduces overheads
  - Hypercalls are very expensive

- To achieve a better performance
  - Application optimization
  - Multicore Hypervisor
  - Adapting compilers to the architecture

Thank you for your attention!