OpenMP Performance on Virtual Machines

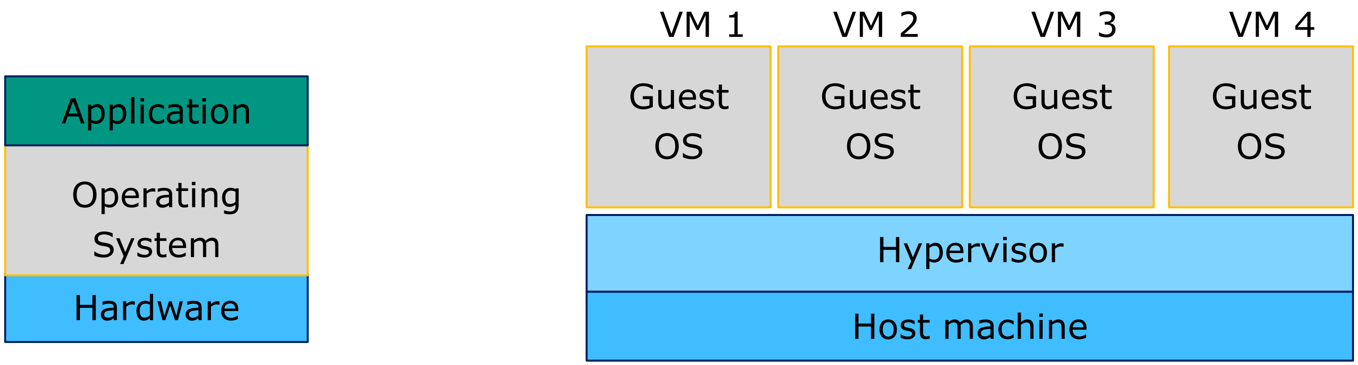
Jie Tao
Karlsruhe Institute of Technology

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

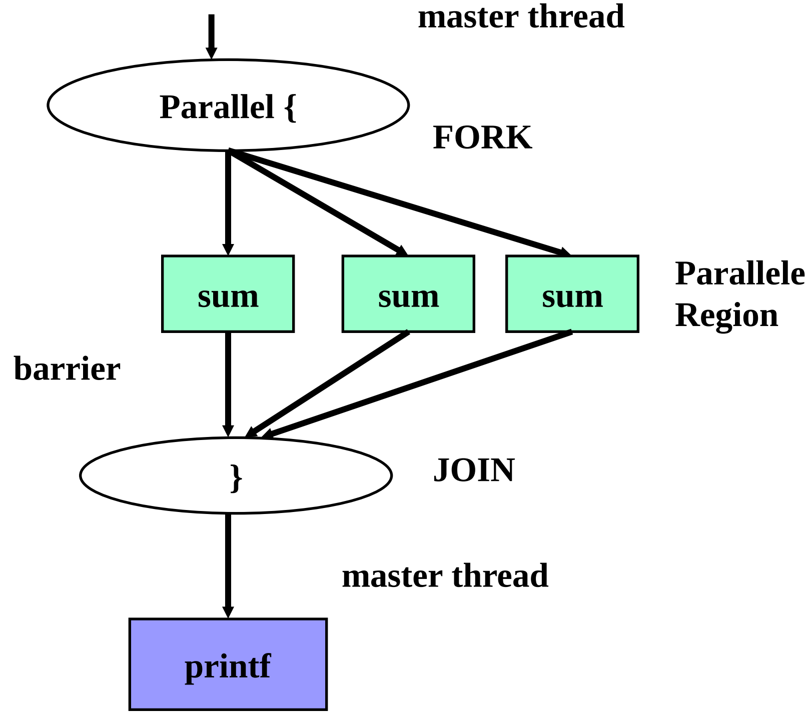
```
#include <stdio.h>
#define LAST 1000
int main()
{
  int i, sum = 0;
#pragma omp parallel for
    reduction(+:sum)
  for ( i = 1; i <= LAST; i++ )
    { sum += i;}
    printf("sum = %d\n", sum);
}
```

Compiling:

gcc –fopenmp -o example example.c

Execution

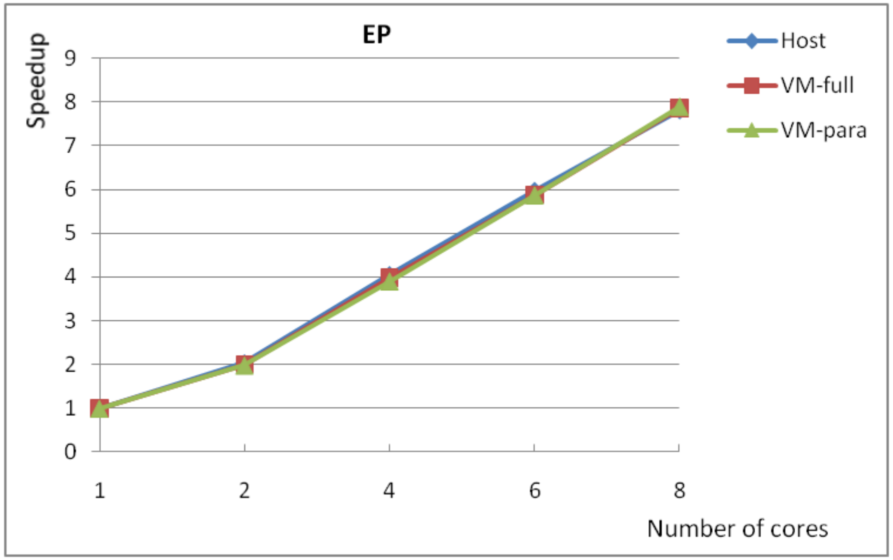
```
$ export OMP_NUM_THREADS=2
$ ./example
```

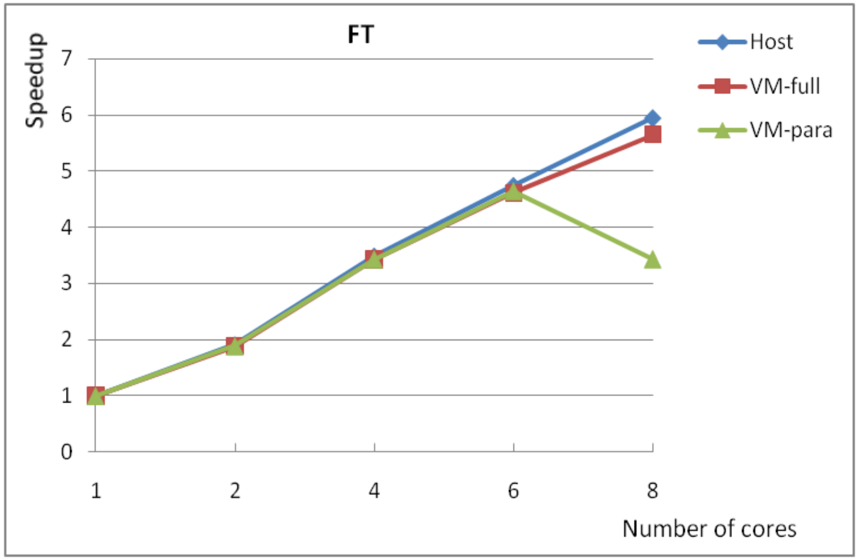


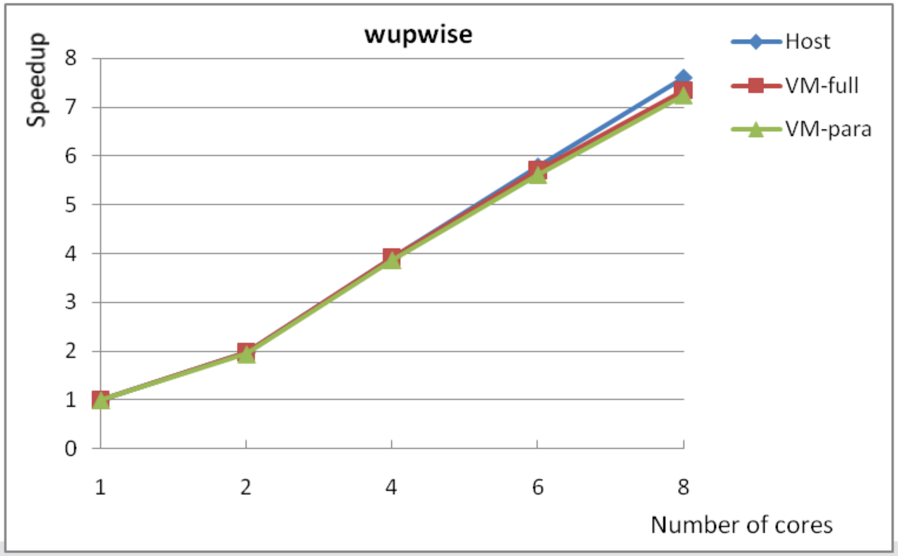
Experimental Setup

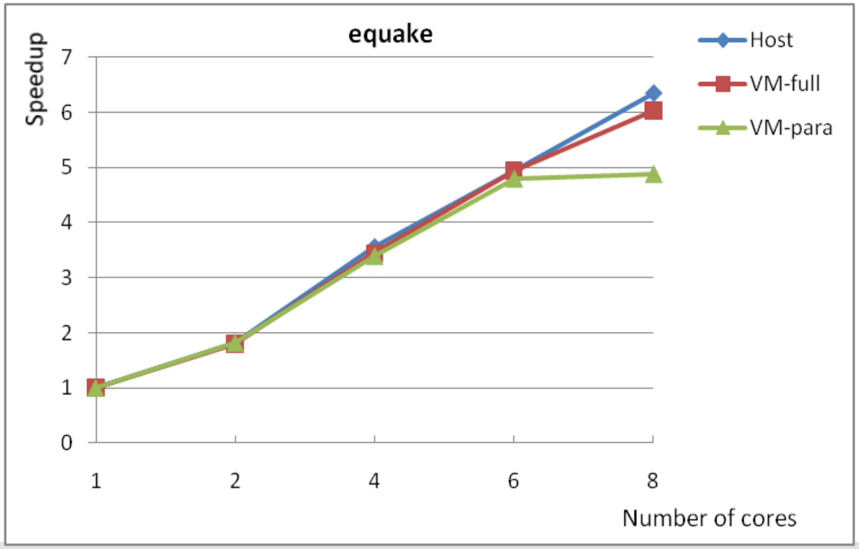
- Host machine
  - AMD multicore
  - Opteron(tm) Processor 2376
  - Scientific Linux
  - Virtualized with xen and KVM
- Virtual machines
  - Hypervisor: xen
  - OS: Debian 2.6.26
  - Compiler: gcc 4.3.2
  - #cores: 1-8
  - Memory: 4GB
- Benchmarks
  - SPEC OpenMP
  - NAS OpenMP
  - OpenMP Microbenchmarks

Speedup

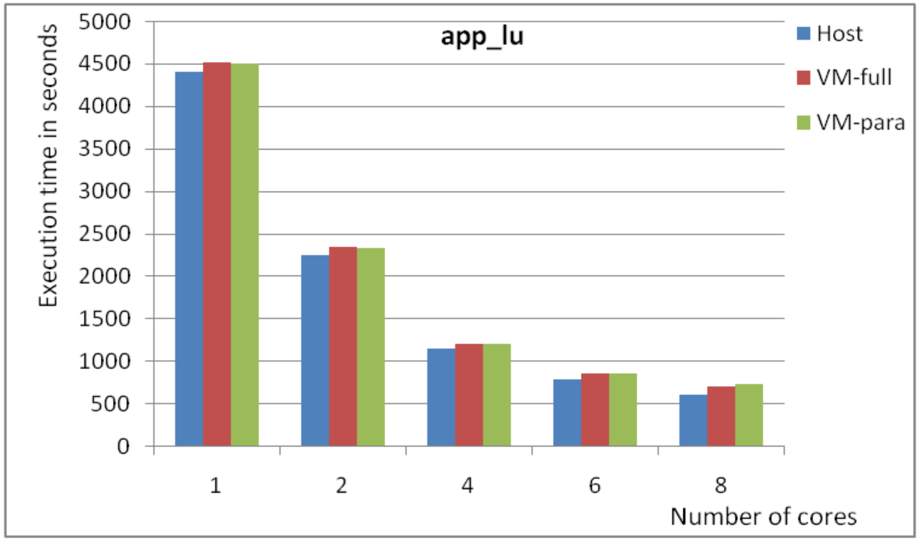

EP
Speedup
0
1
2
3
4
5
6
7
8
9
1
2
4
6
8
Number of cores
Host
VM-full
VM-para

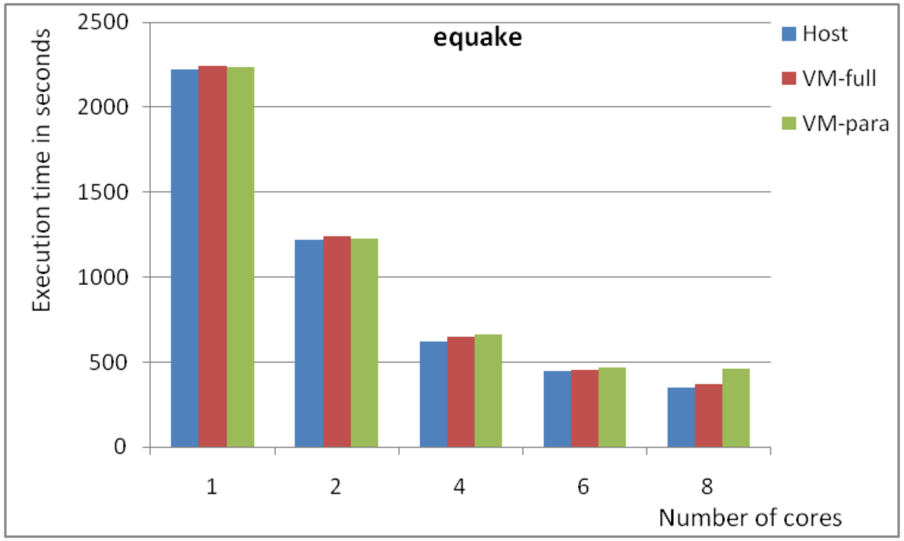

FT
Speedup
0
1
2
3
4
5
6
7
1
2
4
6
8
Number of cores
Host
VM-full
VM-para

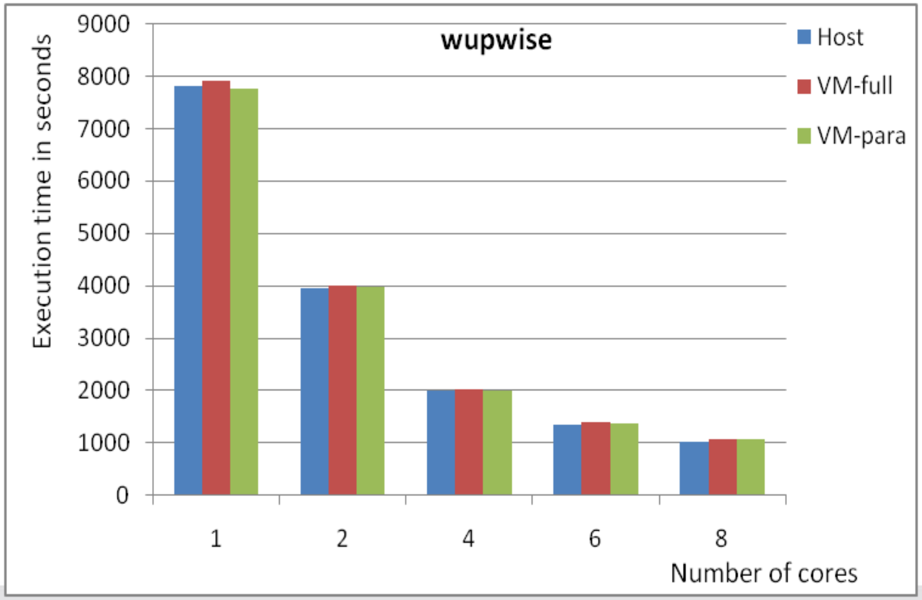

wupwise
Speedup
0
1
2
3
4
5
6
7
8
1
2
4
6
8
Number of cores
Host
VM-full
VM-para

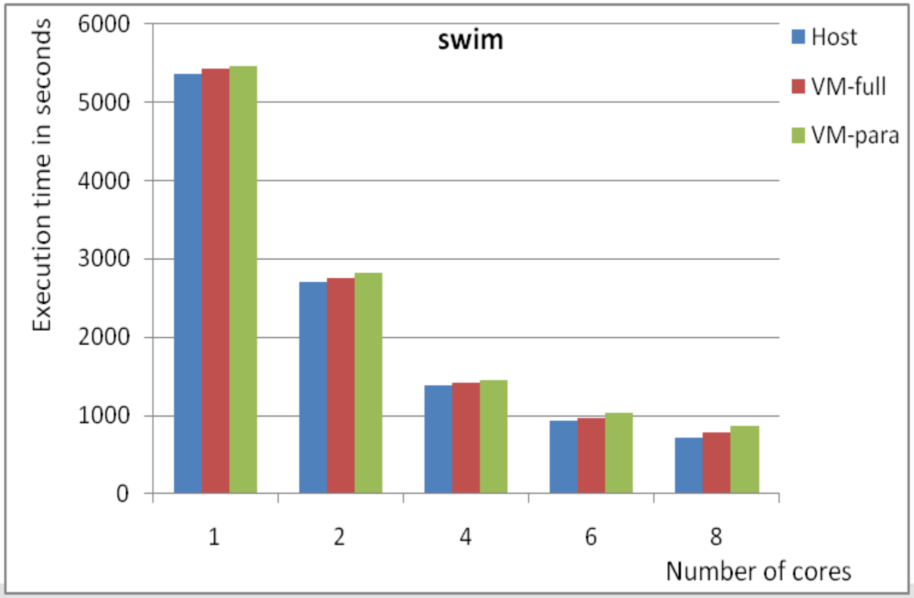

equake
Speedup
0
1
2
3
4
5
6
7
1
2
4
6
8
Number of cores
Host
VM-full
VM-para

Execution Time - SPEC

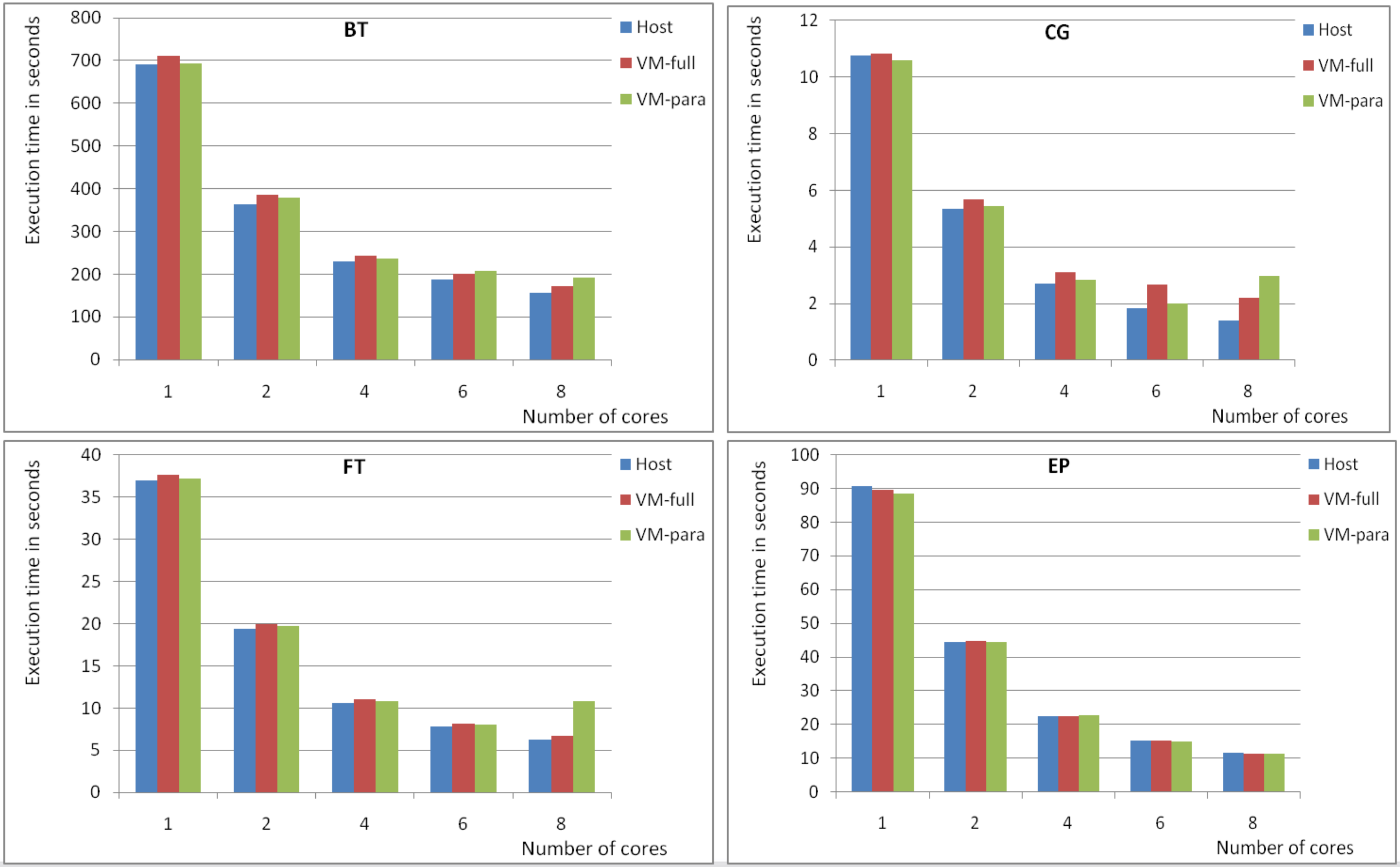




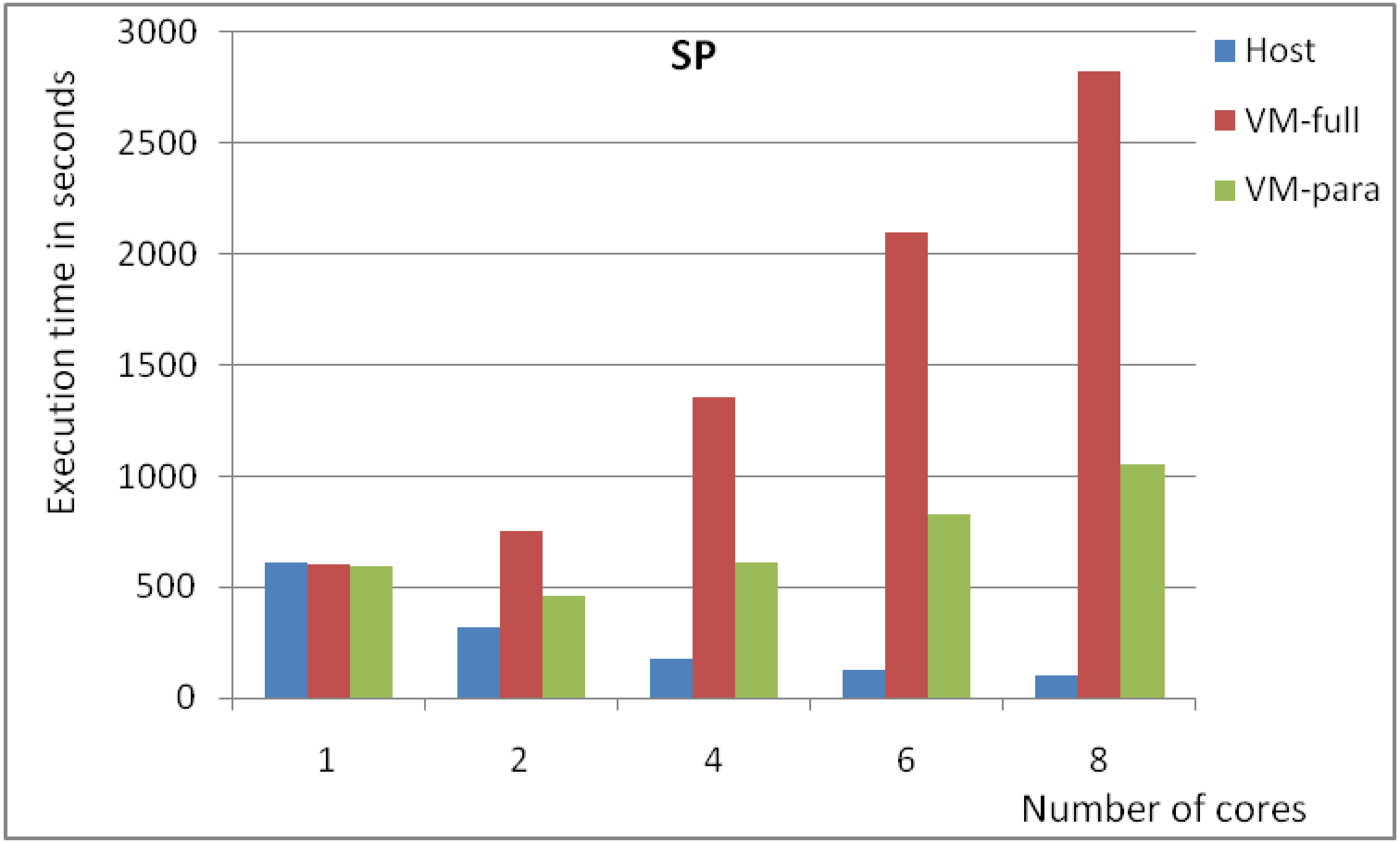




Execution Time - NAS



Execution Time – NAS/SP



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

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

Region Overview of NAS Applications

	PARALLEL	LOOP	SINGLE	BARRIER	CRITICAL	MASTER
BT	2	54	0	0	0	2
FT	2	6	5	1	1	1
CG	2	22	12	0	0	2
EP	1	1	0	0	1	1
SP	2	69	0	3	0	2

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

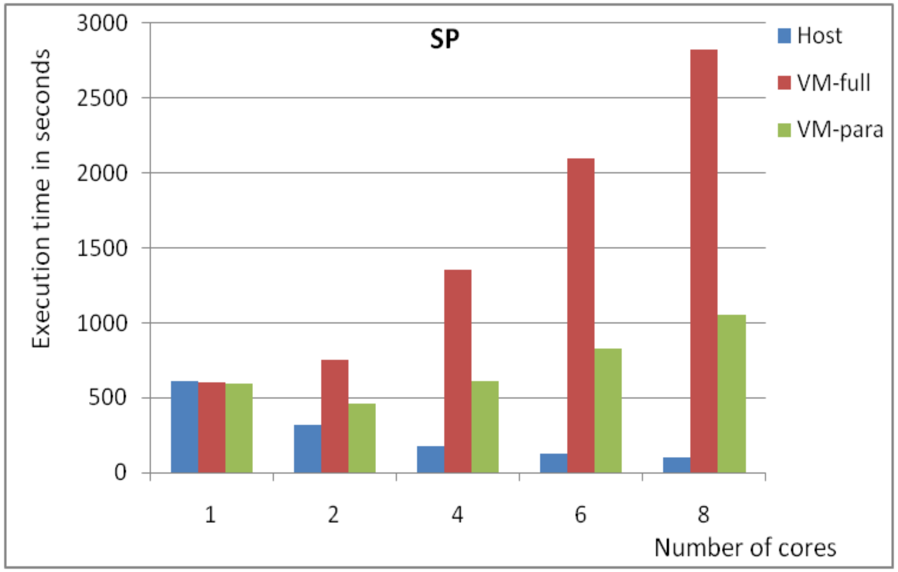
TID	execT	execC	bodyT	exitBarT
0	310.60	1541444	11.24	289.41
1	310.50	1541444	11.22	289.35
2	310.44	1541444	11.33	289.12
3	310.26	1541444	11.22	289.14
4	310.85	1541444	11.26	289.68
5	310.82	1541444	11.24	289.62
6	311.10	1541444	11.17	289.99
7	311.14	1541444	10.92	290.48
SUM	2485.71	12331552	89.60	2316.76

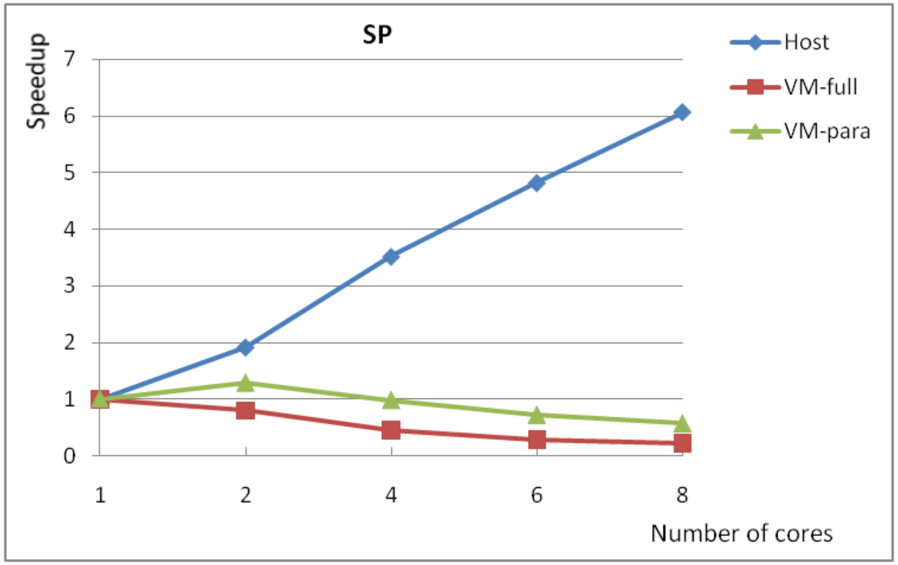
Optimization

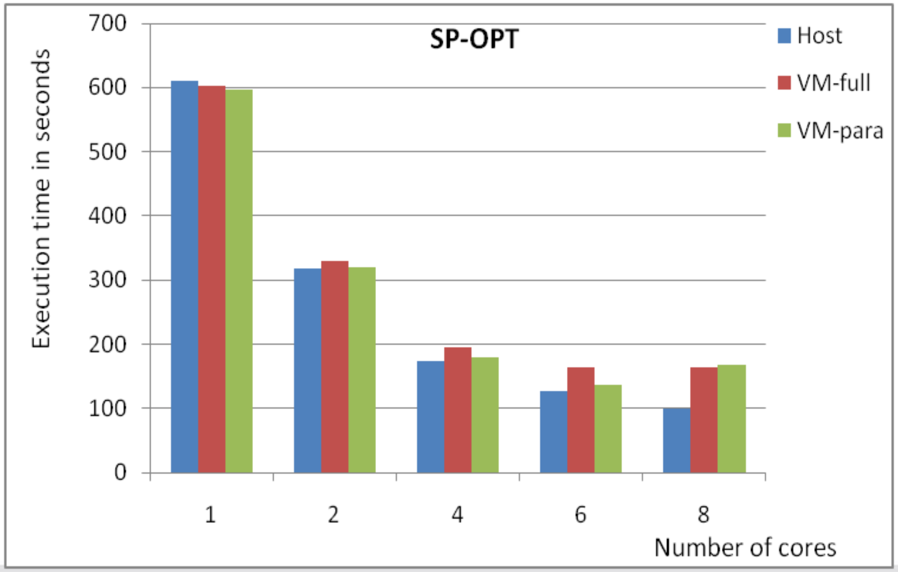
```
for (j = 1; j <= grid_points[1]-2; j++) {
    for (k = 1; k <= grid_points[2]-2; k++) {
#pragma omp for
      for (i = 0; i <= grid_points[0]-1; i++) {
               ru1 = c3c4*rho_i[i][j][k];
               cv[i] = us[i][j][k];
                rhon[i] = max(dx2+con43*ru1,
                     max(dx5+c1c5*ru1,
                         max(dxmax+ru1,
                             dx1)));
      }

#pragma omp for
      for (i = 1; i <= grid_points[0]-2; i++) {
               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;
      }
    }
  }
```

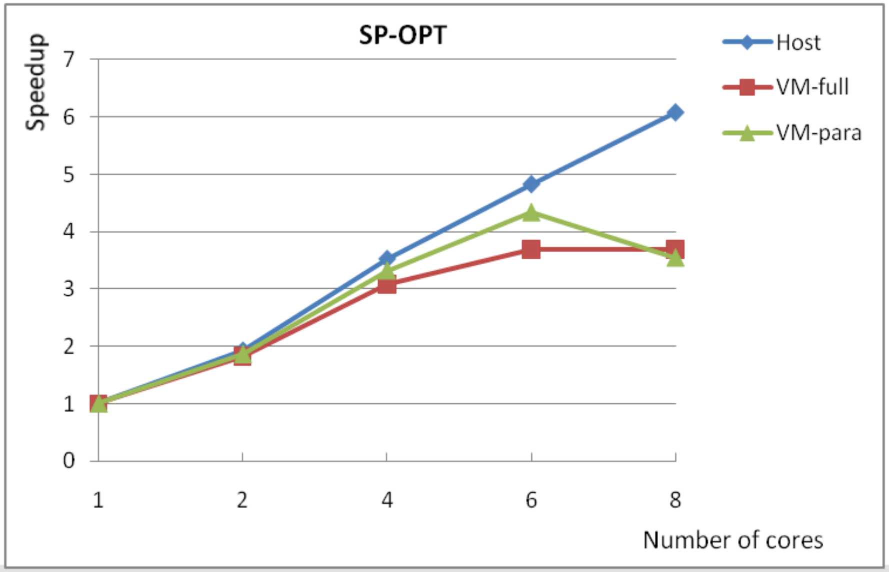
```
#pragma omp for
 for (j = 1; j <= grid_points[1]-2; j++) {
   for (k = 1; k <= grid_points[2]-2; k++) {
     for (i = 0; i <= grid_points[0]-1; i++) {
       ru1 = c3c4*rho_i[i][j][k];
       cv[i] = us[i][j][k];
       rhon[i] = max(dx2+con43*ru1,
                     max(dx5+c1c5*ru1,
                         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









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!