LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-512249

Understanding Performance of Parallel Scientific Simulation Codes using Open|SpeedShop

K. K. Ghosh

November 9, 2011

SC'11 International Conference for High Performance Computing, Networking, Storage and Analysis
Seattle, WA, United States
November 12, 2011 through November 18, 2011

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Understanding Performance of Parallel Scientific Simulation Codes using Open|SpeedShop

SC'11 NNSA/ASC Booth Presentation

November 17, 2011

Koushik K Ghosh, LLNL

LLNL-CONF-512249

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

Topics For Getting Started

- Simple but useful example
- Typical counters to use
- What tools to use for measurements
- What to do with findings

Typical Counters to Use

- PAPI_TOT_CYC
- PAPI_TOT_INS
- PAPI_FP_OPS
- PAPI_L1_DCA and PAPI_L1_DCM
- PAPI_L2_DCA and PAPI_L2_DCM
- MEM_INST_RETIRED:LOADS:STORES
- LAST_LEVEL_CACHE_REFERENCES,LAST_LEVEL_CACHE_MISSES
- PAPI_VEC_DP

osshwcsamp

- Use osshwcsamp convenience script to deploy openss
- Use PAPI names
- Use Native names
- Use HWCSAMP experiments
- Use multiplexing (up to 6 counters)
- Some counters are not compatible
- Use papi_event_chooser
- Comma separated PAPI event list to osshwcsamp
- osshwcsamp “srun –n288 –N24 smg2000 –n 12 12 12” PAPI_TOT_CYC,PAPI_TOT_INS,PAPI_FP_OPS

Viewing hwcsamp Data

Open|SpeedShop

File Tools Help

HWCSamp Panel [1]

Process Control

Run Cont Pause Update Terminate

Status: Process Loaded: Cli

Exclusive Time

Multiple HWC Event Counts

Stats Panel [1] ManageProcessesPanel [1]

Showing Functions Report:

View/Display Choice: Functions Statements Linked Objects

Executables: smg2000 Host: localhost.localdomain Processes/Ranks/Threads:(2) 0 ...

% of CPU Time	Exclusive CPU time in sec	% of CPU Time	PAPI_TOT_CYC	PAPI_FP_OPS	Function (defining location)
44.697833523	3.920000000	44.697833523	11772604888	11984[illegible]900	hypre_SMGResidual (smg2000: smg_
28.620296465	2.510000000	28.620296465	7478131309	812850606	hypre_CyclicReduction (smg2000: cy
3.534777651	0.310000000	3.534777651	915610917	48863259	opal_progress (libopen-pal.so.0.0.0)
3.420752566	0.300000000	3.420752566	910260309	100529525	hypre_SemiRestrict (smg2000: semi_
3.306727480	0.290000000	3.306727480	874155835	48509938	mca_btl_sm_component_progress (li
2.964652223	0.260000000	2.964652223	781833036	98664505	hypre_SemiInterp (smg2000: semi_i
other					

Command Panel

openss>>

What Counters are Supported?

- papi_avail –a # derived
- papi_native_avail # native
- Example output:

The following correspond to fields in the PAPI_event_info_t structure.

Name	Code	Deriv	Description(Note)
PAPI_L1_DCM	0x80000000	No	Level 1 data cache misses
PAPI_L1_ICM	0x80000001	No	Level 1 instr cache misses
PAPI_L2_DCM	0x80000002	Yes	Level 2 data cache misses
PAPI_L2_ICM	0x80000003	No	Level 2 instr cache misses
PAPI_L1_TCM	0x80000006	Yes	Level 1 total cache misses
PAPI_L2_TCM	0x80000007	No	Level 2 total cache misses

What Counters are Supported – cont..

- **papi_native_avail # native events**
- **Example output:**
- **MEM_INST_RETIRED - Memory instructions retired**
- **:LOADS - Instructions retired which contains a load (Precise Event)**
- **:STORES - Instructions retired which contains a store (Precise Event)**
- **:LATENCY_ABOVE_THRESHOLD - Memory instructions retired above programmed clocks, minimum value threshold is 4**

- **Typical use: MEM_INST_RETIRED:LOADS:STORES**

About your System

- papi_mem_info # great utility!!!
  - L1 Data Cache Total size: 32 KB Line size: 64 B #Lines: 512 Associativity: 8
  - L2 Unified Cache: Total size: 256 KB Line size: 64 B #Lines: 4096 Associativity: 8
  - L3 Unified Cache: Total size: 12288 KB Line size: 64 B #Lines: 196608 Associativity: 16
- cat /proc/cpuinfo
- cat /proc/meminfo
- Intel Xeon Hex-core Westmere @ 2.8 GHz
  - 24GB/node
  - 1944 nodes (Peak perf 261.3 Tflops)
  - 12 cores / node (2 sockets / node)
  - QLogic QDR 4x Infiniband
  - 12 MB shared L3 cache

A Simple Example of Use of Perf Counters, Calibration and Sanity Check

Method 1 – Loop structure IJK

```
void  method1 () {
    int i,j,k;
    f or (i=0; i < N; i++){
            for (j=0; j<N; j++){
                double sum = 0.0;
                for (k=0; k<N; k++){
                    sum += a[i][k] * b[k][j];
                }
                c[i][j] = sum;
            }
        }
}
```

Horrible non-unit stride for b[k][j]

Method 2 – Loop structure IKJ

```
void mxm_IKJ () {
    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];
                }
            }
        }
}
```

- a[i][k] : scalar constant
- b[k][j] & c[i][j] : stride-1 vectors

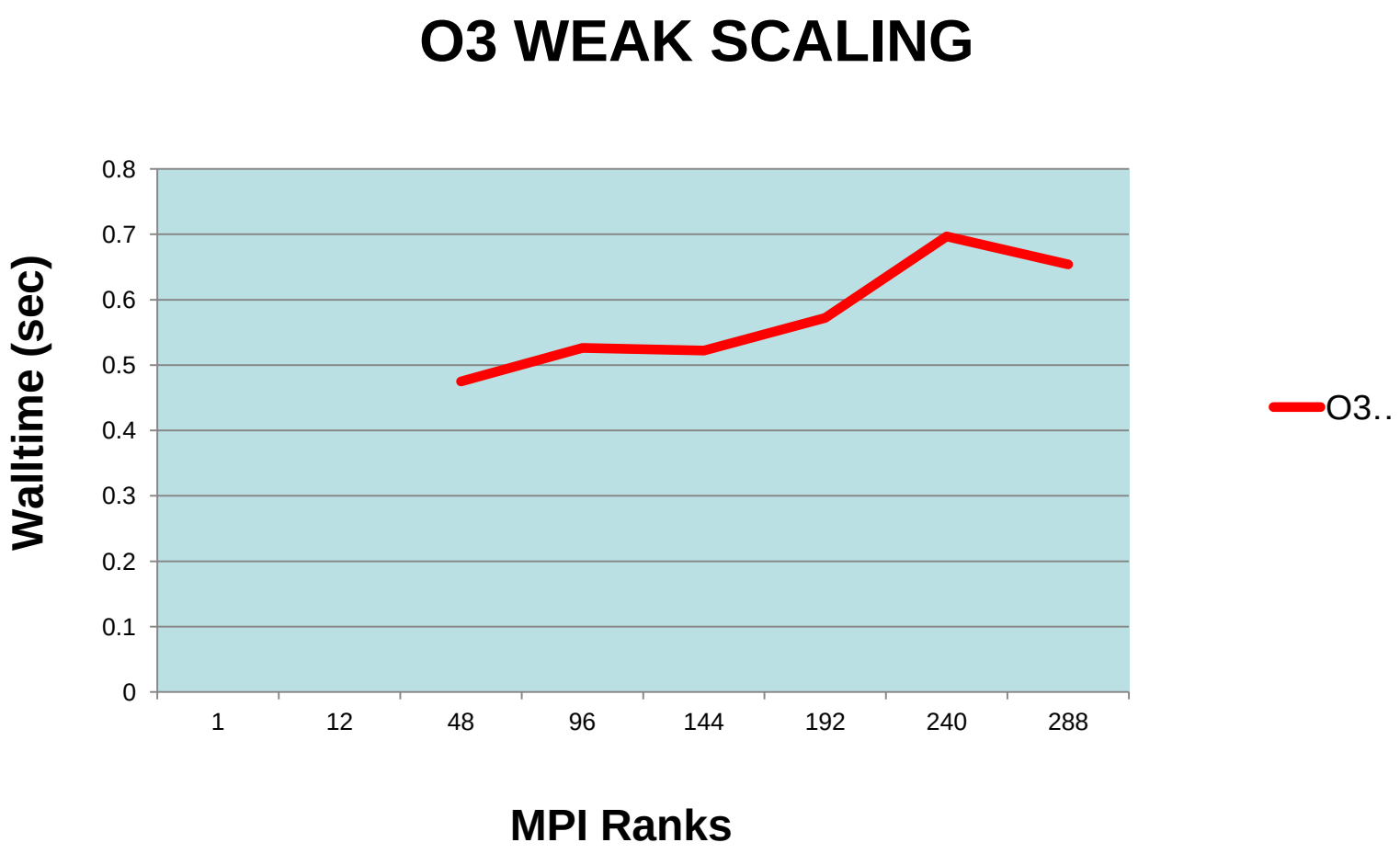
Direct Comparison – MXM Method1 vs. Method2

		Method1	Method2	Ratio
Time		37.5	3.8	10
PAPI_TOT_CYC		115712528500	11901173086	1
PAPI_TOT_INS		47908235122	24656184730	2
PAPI_FP_OPS		27021790436	9860479761	3
PAPI_VEC_DP		125330718	9862200189	0.01
PAPI_L1_DCM		22978703	11512726	2
PAPI_L2_DCA		21851296	11535473	2
PAPI_L2_DCM		297942	5704520	
LLC_REFERENCES		361967	6733426	
LLC_MISSES		17623(4.5%)	95987(1.4%)	
MEM_INST_RETIRED		15167409545	13693987463	
PAPI_TLB_DM		5012430483	8600677	600

Try Performance Analysis of Large Parallel Code

- About the Code
  - Semi-coarse Multi Grid (SMG 2000)
  - Weak Scaling is being studied
  - Memory Intensive
  - Sparse Linear Algebra
  - Paucity of Dense Linear Algebra

O3 Weak Scaling



Initial Code Cache Performance

	Total Instructions	Total Access	Miss	Miss/Access	Miss/Instr
L1	18560879220	NA	222820741		0.01
L2	18560879220	229935543	43552498	0.30	0.002
L3	18560879220	65489269	19778060	0.30	0.001
TLB	18560879220		3211186		<0.001

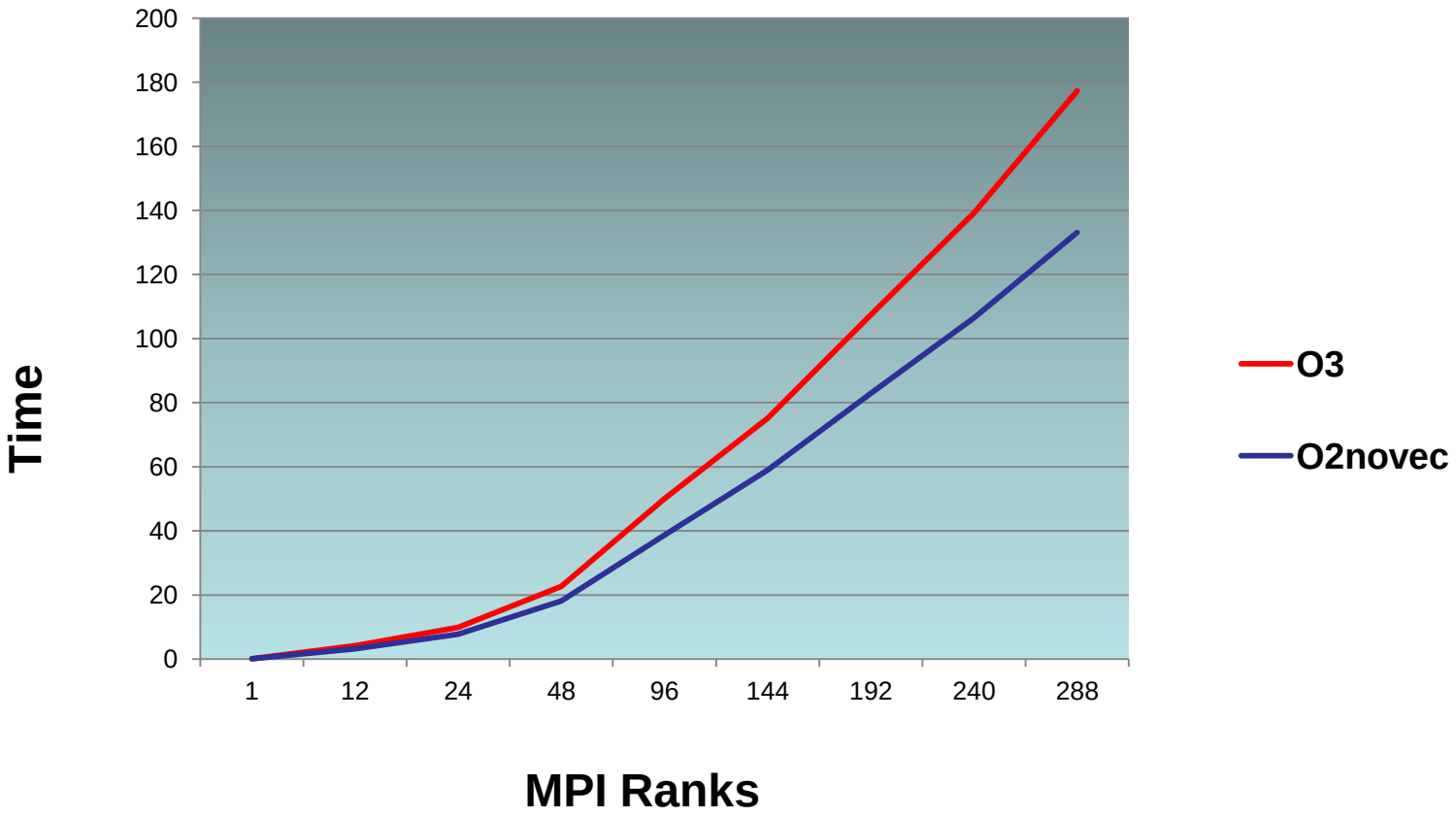
Use another compiler

	Compiler1	Compiler2	
TIME	167.26	144.3	
TOT_CYC	5.0329E+11	4.33517E+11	
TOT_INS	7.12E+11	6.05E+11	
FP_OPS	8.43E+09	1.34E+10	
FP_RATE	14.5198	26.81441	
VEC_OPS	54 G	29 M	

Lesson Learnt

- Compiler2 is better
- Vectorizing better
- Try less aggressive optimization (-O2) ??
- Try a no-vector option ??

Improvement due to noVector



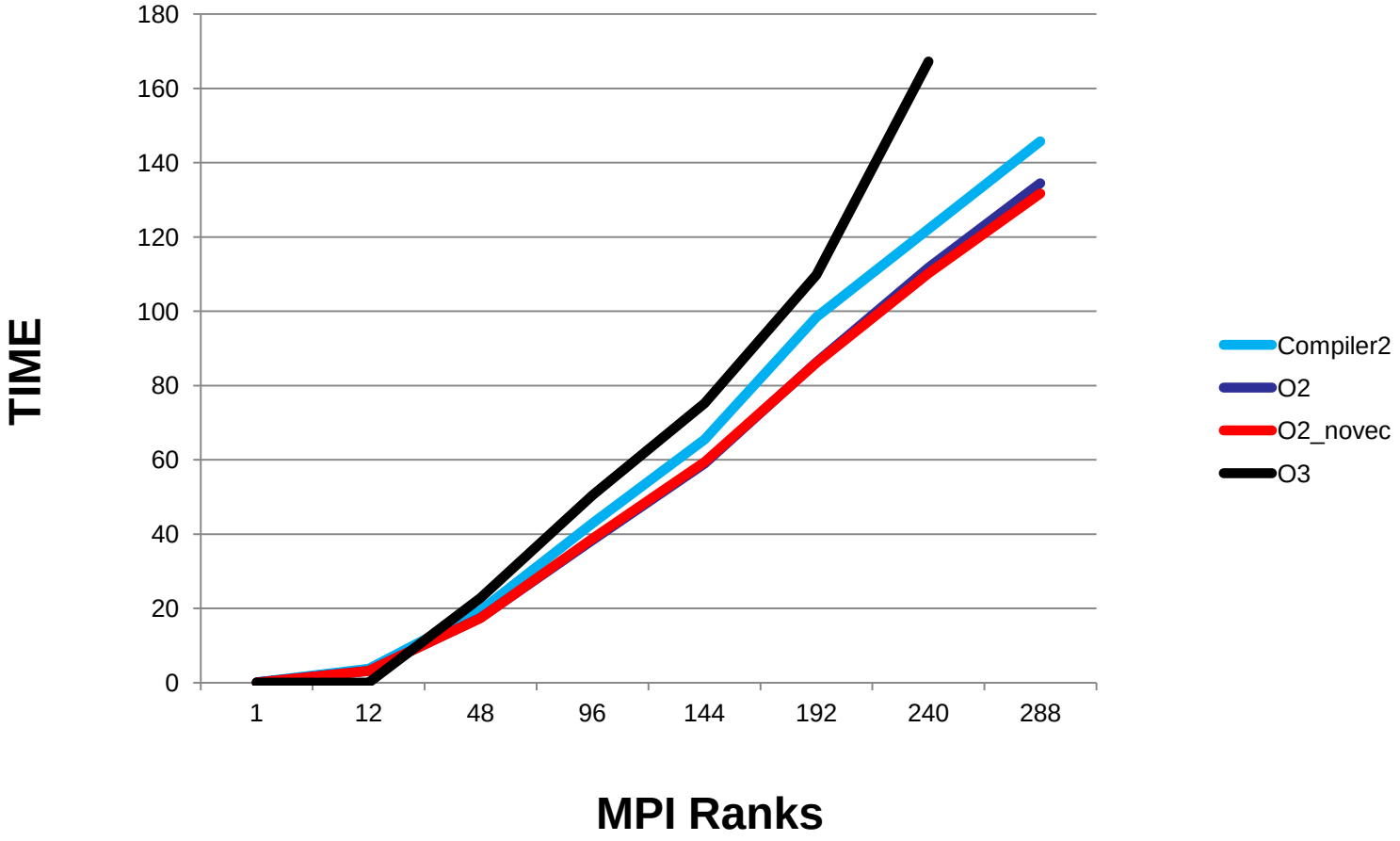
Compiler2 performance

#MPI ranks	Compiler1	Compiler2	
1	na	0.11	
12	na	3.76	
48	22.79	19.45	**1.17**
96	50.52	42.92	**1.17**
144	75.2	65.46	**1.15**
192	109.83	98.4	
240	167.26	122.12	
288	188.41	145.74	

Performance of no-vector code

#MPI ranks	O3_vector	Compiler2	O2	O2_novector
1	na	0.11	0.08	0.07
12	na	3.76	3.21	3.17
48	22.79	19.45	17.46	17.41
96	50.52	42.92	38.43	38.79
144	75.2	65.46	59.01	59.41
192	109.83	98.4	86.3	86.09
240	167.26	122.12	111.74	110.22
288	188.41	145.74	134.44	131.69

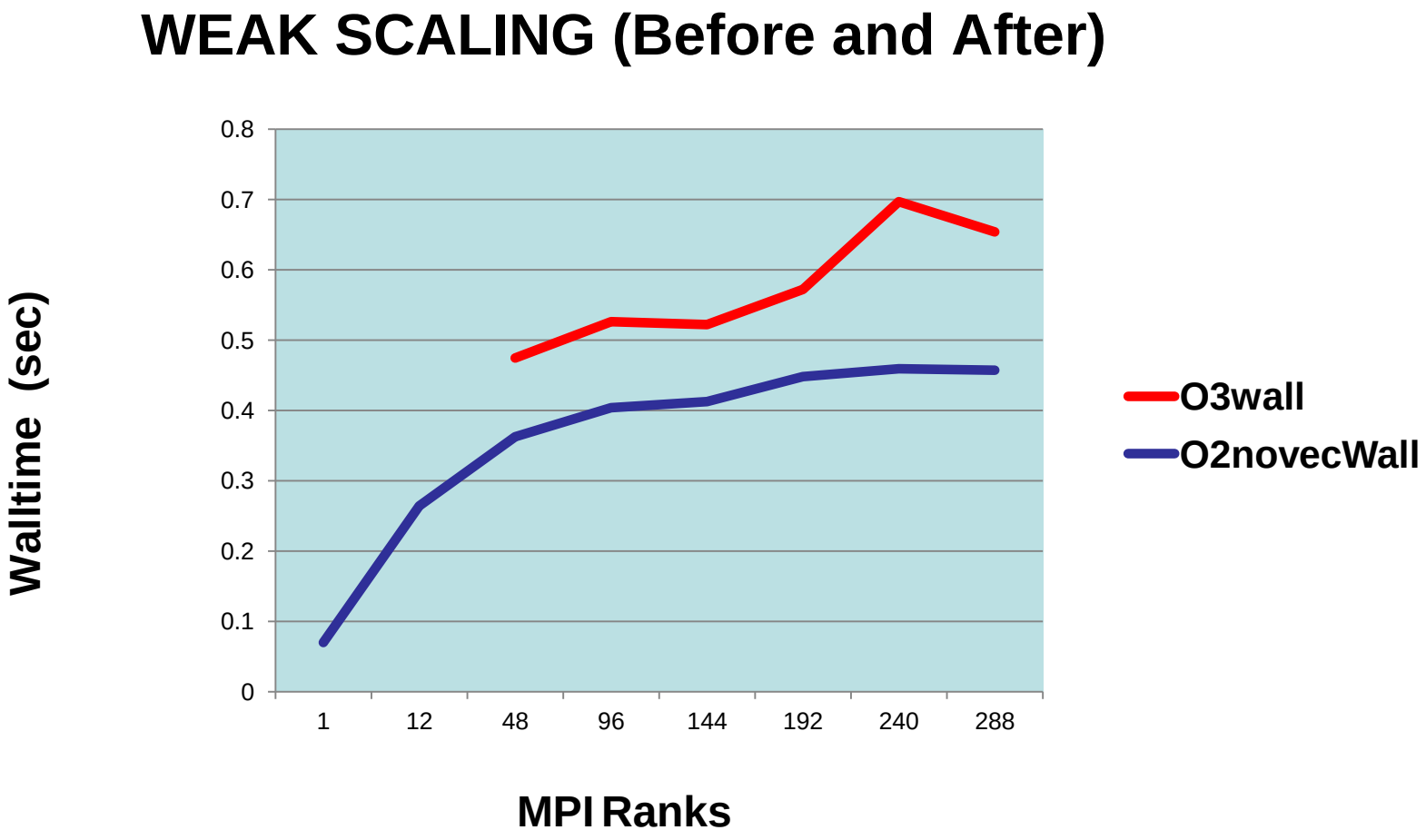
Comparison of successive improvements



Improvement O3 -> O2-noVector

MPI	O3	O2novec	Speedup
1	0.12	0.08	1.50
12	4.14	3.22	1.29
24	9.94	7.77	1.28
48	22.67	18.1	1.25
96	50.03	38.63	1.30
144	75.22	58.99	1.28
192	107.47	82.96	1.30
240	139.22	106.44	1.31
288	177.33	133.09	1.33

Weak Scaling Got Better



Conclusions

- Open SpeedShop's (OSS) is convenient to use for large, parallel, scientific simulation codes
- Large codes benefit from uninstrumented execution
- Many experiments can be run in a short time – might need multiple shots e.g. usertime for caller-callee, hwcsamp for HW counters
- Decent idea of code's performance is easily obtained
- Statistical sampling calls for decent number of samples
- HWC data is very useful for micro-analysis but can be tricky to analyze

Useful Reference

- **Dan Terpstra, "Basic Performanc Measurement with PAPI and Opteron", University of Tennessee, Innovative Computing Lab Seminar.**

  http://icl.cs.utk.edu/newsletter/presentations/2008/Dan-basic-performance-measurement-with-PAPI-07-08.pdf
- **http://icl.cs.utk.edu/papi/**

Acknowledgement

- Jim Galarowicz, Krell Institute
- Don Maghrak, Krell Institute