Efficiency and Scalability of an Explicit Operator on an IBM POWER4 System

Michael Frumkin
NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA 94035-1000
frumkin@nas.nasa.gov

August 26 2002

NAS-02-008

Abstract

We present an evaluation of the efficiency and the scalability of an explicit CFD operator on an IBM POWER4 system.

1 Introduction

The POWER4 architecture [6] exhibits a common trend in HPC architectures: boosting CPU processing power by increasing the number of functional units, while hiding the latency of memory access by increasing the depth of the memory hierarchy. The overall machine performance depends on the ability of the caches-buses-fabric-memory to feed the functional units with the data to be processed.

In this study we evaluate the efficiency and scalability of one explicit CFD operator on an IBM POWER4. This operator performs computations at the points of a Cartesian grid and involves a few dozen floating point numbers and on the order of 100 floating point operations per grid point. The computations in all grid points are independent.

Specifically, we estimate the efficiency of the RHS operator (SP of NPB) on a single processor as the observed/peak performance ratio. Then we estimate the scalability of the operator on a single chip (2 CPUs), a single MCM (8 CPUs), 16 cpus, and the whole machine (32 CPUs). Then we perform the same measurements for a cache-optimized version of the RHS operator.

For our measurements we use the HPM (Hardware Performance Monitor) counters available on the POWER4. These counters allow us to analyze the obtained performance results.

2 Experimental Setup

We used the OpenMP version of the SP NAS Parallel Benchmarks suite (PBN-3.0b4) [3]. We ran SP.A on a 1.3 GHz, 32-processor POWER4 machine (ibm02), provided by IBM to the NAS Division of NASA Ames Research Center. We compiled SP.A with the following options:

- `F77 = xlf_r`
- `FFLAGS = -O4 -qsmp=omp -qnosave -qhot -qtune=pwr4`
- `FLINK = xlf_r`
- `FLINKFLAGS = -qsmp=omp`

We analyzed the performance of explicit operators as defined in the `compute_rhs` and `txinvr` subroutines of the SP benchmark. These subroutines consume about 43% of the total execution time of the SP.A benchmark on the IBM POWER4 machine. In the original version, we inserted calls to start/stop the HPM [4] counters at the entry/exit of the `compute_rhs` and `txinvr` subroutines. In the optimized version, we inserted calls to start/stop the HPM counters at the entry/exit of the merged `compute_rhs/txinvr` subroutines.

3 Experimental Results

We estimate the efficiency of the subroutine on a single processor by using some counters of groups 15 and 60, see Table 1.

Table 1: **The Floating Point Efficiency (FP-efficiency) of the compute˙rhs/txinvr subroutine. Here the FP-efficiency is defined as the ratio of the number of FPU instructions executed by the program to the maximum number of FPU instructions that a single processor is capable of executing during the program WCT.**

Counter	Value
Wall Clock Time (WCT) (sec)	58.9
`PM_FPU_FDIV` (FPU executed FDIV instr.)	200.7E+6
`PM_FPU_FMA` (FPU executed multiply-add instr.)	18.0E+9
`PM_FPU_ALL` (FPU executed add, mult, sub, cmp, or sel instr.)	11.5E+9
`PM_CYC` (Processor cycles)	76.4E+9
FP-efficiency	15.5 %

Here we define the single processor FP-efficiency as the ratio of the number of FPU instructions executed by the program to the maximum number of FPU instructions that a single processor is capable of executing during the program WCT. Hence the FP-efficiency equals (11.5E+9+2*18.0E+9)/(4*1.3E+9*58.9)

= 15.5%. This FP-efficiency is comparable to the 20% efficiency of CFD codes on the SGI Origin 2000 [5]. It is well known that the main bottleneck of cache based systems is feeding the processor with data. We have shown that on Origin systems data reuse, nest fuse, loop interchange, and removal of auxiliary arrays improve performance of the `compute_rhs` subroutine by 40% see Table 2 in [2]. Here we also merge the `compute_rhs` and `txinvr` subroutines.

In the following we study the POWER4 memory subsystem using counters of groups 5 and 59:

- `PM_CYC` (Processor cycles)
- `PM_LSU_LMQ_SRQ_EMPTY_CYC` (Cycles LMQ and SMQ empty)
- `PM_DATA_FROM_MEM` (Data loaded from memory)
- `PM_DTLB_MISS` (Data TLB misses)
- `PM_DATA_FROM_L3` (Data loaded from L3)
- `PM_DATA_FROM_L35` (Data loaded from L3.5)
- `PM_DATA_FROM_L2` (Data loaded from L2)
- `PM_DATA_FROM_L25_SHR` (Data loaded from L2.5 shared)
- `PM_DATA_FROM_L25_MOD` (Data loaded from L2.5 modified)
- `PM_DATA_FROM_L275_SHR` (Data loaded from L2.75 shared)
- `PM_DATA_FROM_L275_MOD` (Data loaded from L2.75 modified)
- `PM_LD_MISS_L1` (L1 Data cache load misses)
- `PM_ST_MISS_L1` (L1 Data cache store misses)
- `PM_LD_REF_L1` (L1 Data cache load references)
- `PM_ST_REF_L1` (L1 Data cache store references)

Note, since the POWER4 uses the enhanced MESI protocol [6], then in addition to direct cache traffic events accumulated in the `PM_DATA_FROM_L3` and `PM_DATA_FROM_L2` counters, it has cross-L3 and cross-L2 traffic events. The counter `PM_DATA_FROM_L35` accumulates the traffic volume between a processor and an L3 cache in other MCMs. The counters `PM_DATA_FROM_L25_SHR` and `PM_DATA_FROM_L25_MOD` accumulate the traffic volume between a processor and an L2 cache in the same MCMs, while the counters `PM_DATA_FROM_L275_SHR` and `PM_DATA_FROM_L275_MOD` accumulate the traffic volume between a processor and an L2 cache in other MCMs.

In addition to these counters, in our plots, we have included WCT as reported by the HPM for each instrumented section of the code. The subroutine execution

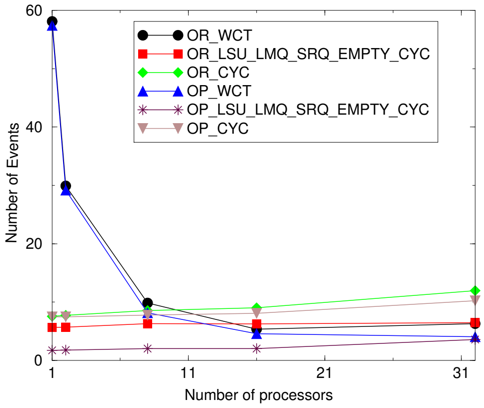

Figure 1: The performance of the two versions of the `compute_rhs` and `txinvr` subroutines. WCT is in seconds, `LSU_LMQ_SRQ_EMPTY_CYC` in 10^9, events and CYC is in 10^{10} events. Here and below, we replace the counter prefix PM with OR for the original program and OP for the optimized one.

Table 2: **The performance of the two versions of the `compute_rhs/txinvr` Subroutine.**

Counter/nprocs	1	2	8	16	32
`OR_WCT` (sec)	58.09	29.92	9.81	5.32	6.27
`OP_WCT` (sec)	57.42	29.16	8.14	4.57	4.06
`OR_CYC`	7.53	7.73	8.52	9.02	11.94
`OP_CYC`	7.44	7.47	7.76	8.06	10.21
`OR_LSU_LMQ_SRQ_EMPTY_CYC`	5.66	5.69	6.30	6.25	6.47
`OP_LSU_LMQ_SRQ_EMPTY_CYC`	1.72	1.75	2.00	2.03	3.55

times, total number of cycles, and number of cycles when the Load/Store Unit (LSU) queues are empty are shown in Figure 1 and Table 2.

This plot shows that the single chip execution times (1 and 2 processors) are almost identical for both versions of the code. On 8-32 processors, the optimized code is 12%-30% faster than the original code. This partially results from a bigger overhead (see CYC curve) and longer idling (`LSU_LMQ_SRQ_EMPTY_CYC` curve) of the original code.

The memory traffic volume and the TLB miss numbers are shown in Figure 2, while a comparison of L3 traffic is shown in Figure 3. Figure 2 shows expected reduction in main memory traffic (up to a factor of 2.3 on a single MCM) and TLB misses (up to a factor of 3) for the optimized code. Figure 3 shows that the optimized code obtains more data from L3 than the original code because it is necessary to compensate for the smaller volume of main memory traffic. This figure also shows that the volume of the cross MCM L3 traffic in the optimized code on the 32 processors is larger than that in the original code.

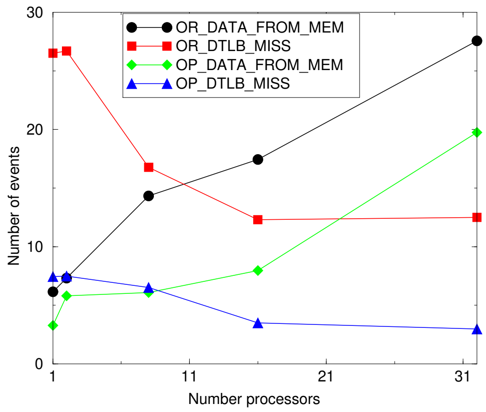

Figure 2: The performance of the two versions of the `compute_rhs` and `txinvr` subroutines. `DATA_FROM_MEM` in GBytes, `DTLB_MISSES` are in 10^6 events.

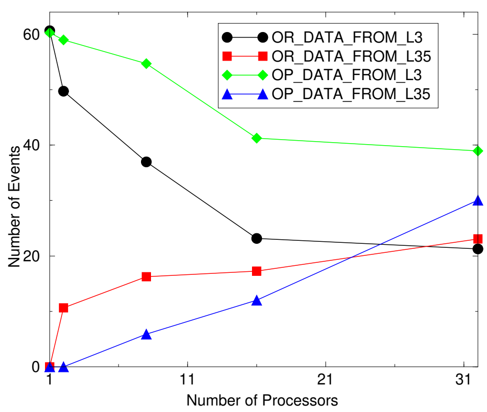

Figure 3: A comparison of L3 traffic volume (GB) for the original and the optimized versions of the subroutine.

The graphs of Figures 4 and 5 show that both cross chip and cross MCM L2 traffic of the original code are about 25-100% more intensive than that of the optimized code, except cross MCM L2 shared traffic which is about the same.

The graphs of Figures 6 and 7 show that both codes have about the same number of L1 misses and L1 references.

4 Interpretation of the Results

The results show a correlation between the main memory traffic and the program performance. The shift of the memory traffic to L3 traffic does not improve performance of single chip (1-2 processor) programs. On the other hand, for 8-32 processors, optimization of the memory traffic does improve program performance. At the extreme of 32 processors, the original program runs slower than on 16 processors, while the optimized program runs faster than on 16 processors. The improved data locality reduces both L2 and cross L2 traffic (Figures 4 and 5) which improves overall performance.

5 Conclusions

We have to follow a fine line in drawing conclusions from the limited experiments we have performed on such a complex architecture as the POWER4. Some circumstances of the execution environment were beyond our control, particularly the interference with other users and a layout of the threads across the processors. We minimized side effects of these uncontrollable circumstances

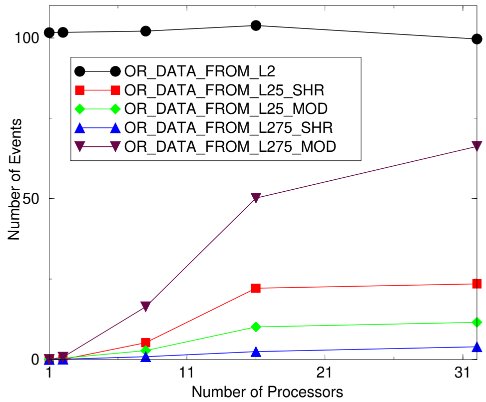

Figure 4: The L2 traffic volume (GB) in the original version of the subroutine. The volume of cross L2 traffic is shown in a 100 times smaller scale (0.01 GB).

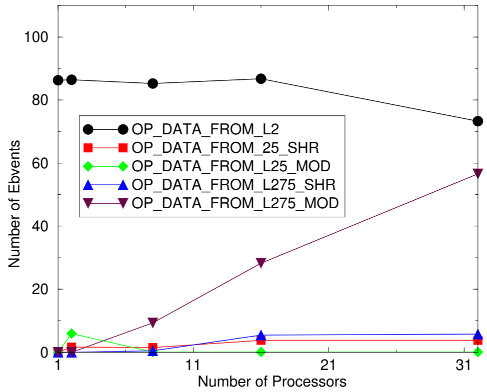

Figure 5: The L2 traffic volume (GB) in the optimized version of the subroutine. The volume of cross L2 traffic is shown in a 100 times smaller scale (0.01 GB).

by monitoring workload with `topas` and reporting the best result of a set of multiple runs of the code.

Our experiments show that the compiler performs good code optimization for single chip (1 and 2 processors) execution. Our attempts to optimize L2 and L1 performance with an elaborate tiling of the operator's iteration space did not yield an essential reduction in the L2 and L1 misses.

On the other hand, the complexity of the MESI protocol [1] and of its enhanced version implemented in the POWER4 system [6] does not allow the compiler to do an optimization of the same quality beyond a single MCM. Our data locality optimizations reduced memory traffic and improved performance by 15% and 20% on 16 and 32 processors, respectively.

Acknowledgment. The author appreciates the guidance of IBM staff members Charles Grassl and Luiz DeRose in the POWER4 architecture, and editorial suggestions by Randy Kaemmerer.

References

[1] David E. Culler. Protocol Design Tradeoffs in Snooping Cache Coherent Multiprocessors. CS 258, Spring 99. U.C. Berkeley. http://www.cs.berkeley.edu/c̃uller/cs258-s99/slides/lec07/sld001.htm.

[2] M. Frumkin, H. Jin, J. Yan. Automation of Data Traffic Control on DSM Architectures. Proceedings of ICCS 2001, Part II, LNCS 2074, pp.771-780.

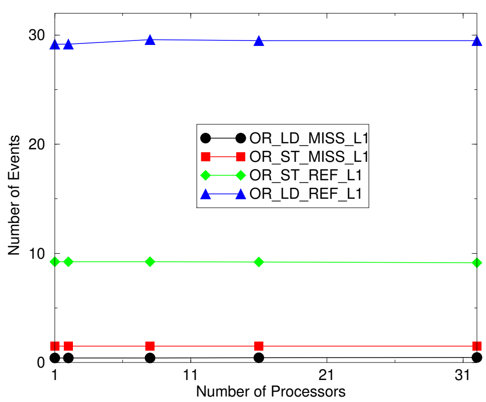

Figure 6: The L1 misses/references in the original version of the subroutine (10^9 events).

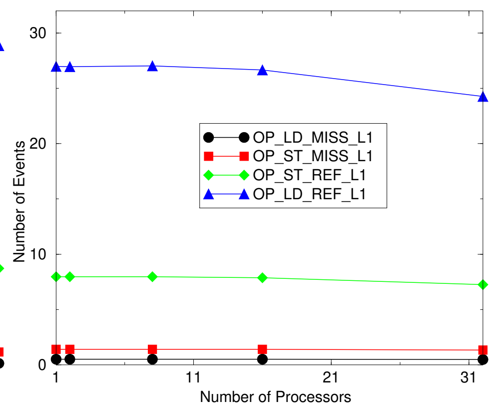

Figure 7: The L1 misses/references in the optimized version of the subroutine (10^9 events).

[3] H. Jin, M. Frumkin, J. Yan. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. NAS Technical Report RNR-99-011, www.nas.nasa.gov.

[4] L. DeRose. The Hardware Performance Monitor Toolkit. In Proceedings of Euro-Par, August 2001, pp. 122-131.

[5] J. Taft. Performance of the Overflow-MLP CFD Code on the NASA Ames 512-CPU Origin System. NAS Technical Report, NAS-00-05, www.nas.nasa.gov.

[6] J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy. POWER4 System Microarchitecture. IBM Journal of Research and Development, Vol. 46, No. 1, 2002. http://www.research.ibm.com/journal/rd/461/tendler.html.