

Porting Applications to the Blue Gene/L Platform

BlueGene/L Optimization Tips

Adapted from Bob Walkup's BNL Presentation

Day 2, Groningen Workshop

Dec 8, 2004

Summary of This Talk

- Characteristics of the hardware
- Getting the most out of IBM XL compilers
- Profiling to identify performance issues
- Using library routines for math kernels

Powerpc-440 Processor

- 32-bit architecture at 700 MHz
- single integer unit (fxu)
- single load/store unit
- special double floating-point unit (dfpu)
- L1 Data cache : 32 KB total size, 32-Byte line size,
- 64-way associative, round-robin replacement
- L2 Data cache : prefetch buffer, holds 16 128-byte lines
- L3 Data cache : 4 MB, ~35 cycles latency
- Memory : 512 MB DDR at 350 MHz, ~85 cycles latency
- Double FPU has 32 primary floating-point registers, 32 secondary floating-point registers, and supports :
- standard powerpc instructions, which execute on fpu0 (fadd, fmadd, fadds, fdiv, ...), and
- SIMD instructions for 64-bit floating-point numbers (fpadd, fpmadd, fpre, ...)
- Floating-point pipeline : 5 cycles
- Floating-point load-to-use latency : 4 cycles

Instruction Latencies and Throughputs

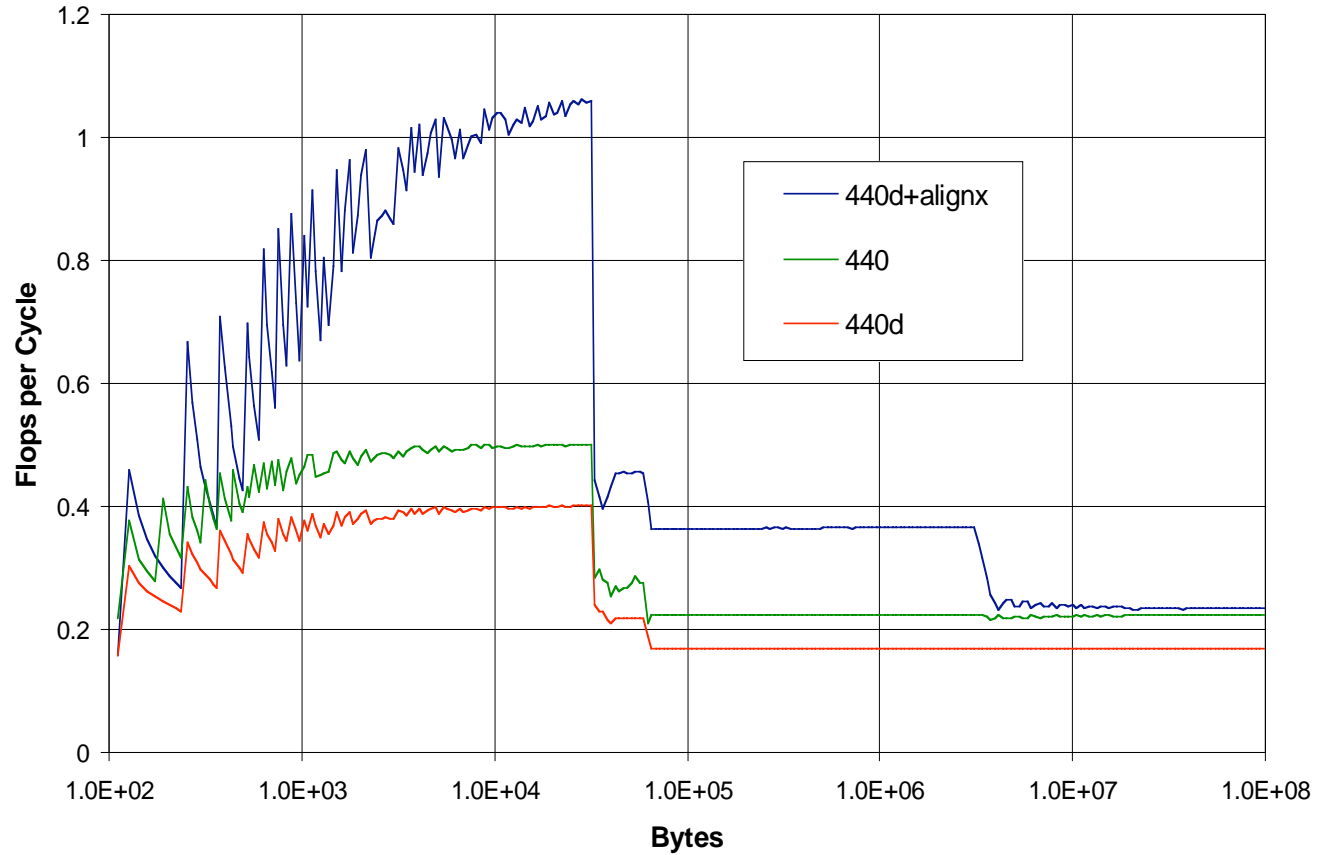
- Instruction latency throughput/cycle
- fadd 5 cycles 1
- fmadd 5 cycles 1
- fpmadd 5 cycles 1
- fdiv 30 cycles 1/30
- Theoretical flop limit = 1 fpmadd per cycle => 4 floating-point ops per cycle.
- Practical limit is often loads and stores.
- No hardware square-root function.
Default sqrt() is from GNU libm.a => ~100 cycles.
- Efficient use of the double-hammer requires 16-byte alignment.
There are quad-word load/store instructions
- (lfpd, stfpd) that can double the bandwidth between L1 and registers. In most applications loads and stores are at least as important as floating-point operations.

```

call alignx(16,x(1))
call alignx(16,y(1))
do i = 1, n
  y(i) = a*x(i) + y(i)
end do

```

BG/L Daxpy Performance



Performance of compiler-generated code is shown.
 -qarch=440 => single FPU code, theoretical limit is 2/3 flops per cycle.
 -qarch=440d => double FPU code, theoretical limit is 4/3 flops per cycle, data in-cache, 2/3 flops per cycle otherwise

Using IBM XL Compilers

Optimization levels:

Default optimization = none (very slow)

-O : good place to start, use with -qmaxmem=64000

-O2: same as -O

-O3 -qstrict : less aggressive, must strictly obey program semantics

-O3: aggressive, allows re-association, will replace division by multiplication with the inverse

-qhot : turns on high-order transformation module, will add vector routines, unless -qhot=novector
check listing: -qreport=hotlist

-qipa : inter-procedure analysis; many suboptions such as: -qipa=level=2

Architecture flags:

-qarch=440 : generates standard powerpc floating-point code

-qarch=440d : will try to generate double FPU code

Recommendation:

On BG/L start with : -g -O -qarch=440 -qmaxmem=64000

Try : -O3 -qarch=440/440d

Try : -O5 -qarch=440d

-O4 = -O3 -qhot -qipa=level=1 -qarch=auto

-O5 = -O3 -qhot -qipa=level=2 -qarch=auto

Some BG/L-Specific Items

For double FPU code generation, 16-byte alignment is required; may need alignment assertions:

Fortran :

```
    call alignx(16,x(1))
    call alignx(16,y(1))
!ibm* unroll(10)
do i = 1, n
    y(i) = a*x(i) + y(i)
end do
```

C :

```
double * x, * y;
#pragma disjoint (*x, *y)
__alignx(16,x);
__alignx(16,y);
#pragma unroll(10)
for (i=0; i<n; i++) y[i] = a*x[i] + y[i];
```

Try : -O3 -qarch=440d -qlist -qsource

Easiest approach to double FPU is to use optimized math library routines.

Profile your Application

BG/L is similar to most risc systems, so profiling can be done on another platform. Standard profiling (prof, gprof) is not yet available on BG/L (as of Oct. 2004), so use IBM Power4 systems for profiling.

Major differences between BG/L and IBM Power4:

BG/L has minimal L2 => BG/L benefits more from L1 cache re-use

BG/L does not have a hardware square root

Mathematical intrinsic routines (exp, log, ...) on BG/L are from GNU libm.a => very slow

Good approach: profile on IBM Power4 / AIX using xprofiler

Use MPI profiling tools – there are many to choose from.

mpi_trace : low overhead, text summary for free (see demo later)

paraver : extensive data analysis capabilities

mpe / jumpshot : standard with MPICH

...

For paraver : <http://www.cepba.upc.es/paraver/>

elapsed time from clock-cycles using freq = 700.0 MHz

```
-----  
MPI Routine      #calls  avg. bytes  time(sec)  
-----  
MPI_Comm_size    6         0.0        0.000  
MPI_Comm_rank    1         0.0        0.000  
MPI_Send         285196    6694.6     35.545  
MPI_Recv         210284    698.5      20.959  
MPI_Probe        81243     0.0        124.980  
MPI_Iprobe       352732    0.0        0.358  
MPI_Bcast        5          4.0        0.000  
MPI_Barrier      10000     0.0        85.153  
MPI_Gather       10002     8.0        0.803  
MPI_Allgather    3         14.7       0.001  
MPI_Allreduce    6         17.3       0.660  
-----
```

MPI task 0 of 512 had the minimum communication time.
total communication time = 268.460 seconds.
total elapsed time = 434.298 seconds.
top of the heap address = 47.293 MBytes.

Message size distributions:

```
MPI_Send      #calls  avg. bytes  time(sec)  
73141         16.0      0.273  
380           45.9      0.004  
786           98.9      0.002  
2325          196.8     0.016  
5768          379.1     0.082  
121925        998.1     22.893  
1023          1600.2    0.011  
643           4079.7    0.112  
511           5721.0    0.019  
77672         10240.0   5.937  
1022          960000.0 6.198
```

-env BGLMPI_EAGER=500 (BG/L default = 10000)
to get adaptive routes for messages of ~1K

Scalar and Vector MASS Routines

Approximate cycle-counts per evaluation on BG/L

	libm.a	libmass.a	libmassv.a
exp	185	64	22
log	320	80	25
pow	460	176	29 - 48
sqrt	106	46	8-10
rsqrt	136	...	6-7
1/x	30	...	4-5