The Apex-MAP Benchmark

Dr. Volker Weinberg
Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften
volker.weinberg@lrz.de

PRACE Workshop “New Languages & Future Technology Prototypes”
LRZ, 1.-2. March 2010
Outline

1. The Apex-MAP Benchmark
2. Modelling LRZ’s Application Mix
3. Validation of ApexMAP: Simulation of 2 Mathematical Kernels on 2 different architectures
4. Summary, References and Acknowledgements
The Apex-MAP Benchmark

Goal:
- A simple synthetic benchmark with tunable hardware indep. parameters that can be used to model the scientific application mix of the HLRB-II would be very useful for the evaluation of new hardware platforms.

Apex-MAP: Application Performance Characterisation Project (Apex) Memory Access Probe (MAP)
https://ftg.lbl.gov/ApeX/ApeX.shtml


- Initial assumption: Performance behaviour can be characterised by a small set of code-specific and architecture-independent performance factors.

- Simulates typical memory access patterns and computational intensities of scientific applications.
Main Parameters of the Apex-MAP Benchmark

- memory access patterns:
  - random access pattern
  - strided access pattern

random access: indexed access using an index buffer $\text{ind}[i]$ whose length is set by the parameter $I$
strided access: access data with stride $S$

- $M$: total size of allocated memory block in which data accesses are simulated
- $L$: number of contiguous memory locations (sub-blocks of length $L < M$) accessed in succession starting at $\text{ind}[i] \rightarrow$ Spatial Locality
- $\alpha$: shape parameter of power distribution function ($0 \leq \alpha \leq 1$), describes temporal reuse of data, $\alpha$ determines the random starting address $\text{ind}[i] \rightarrow$ Temporal Locality
- $C$: computational intensity: own subroutine that runs with peak performance on Itanium
Temporal Locality

\[
\text{ind}[j] = (L \cdot \text{pow}(\text{drand}48(), \frac{1}{\alpha}) \cdot (M/L - 1)) \in [0; M-L]
\]

\[
\alpha \in [0; 1]
\]

\[
\alpha = \begin{cases} 
0 & \text{program accesses a single address repeatedly} \\
1 & \text{uniform random memory access}
\end{cases}
\]

Distribution of \(\text{pow}(\text{drand}48(), \frac{1}{\alpha}) \in [0; 1]\)
random access memory pattern

double *data=(float *) malloc(M*sizeof(double));

for (i = 0; i < I; i++) {
    for (k = 0; k < L; k++) {
        W0 += c0*data[ind[i]+k];
        W0 += compute(C);
    }
}

strided access memory pattern

for (int k = 0; k < M/S; k+=1) {
    W0 += c0*data[k*S];
    W0 += compute(C);
}
double compute(int C){
    double s0,s1,s2,s3,s4,s5,s6,s7;
    s0=s1=s2=s3=s4=s5=s6=s7=0.;
    for(int i=1;i<=C;i++){
        dummy(&s0,&s1,&s2,&s3,&s4,&s5,&s6,&s7);
        s0+=(x[0]*y[0])+(x[0]*y[1])+(x[0]*y[2])+(x[0]*y[3])+
            (x[0]*y[4])+(x[0]*y[5])+(x[0]*y[6])+(x[0]*y[7]);
        s1+=(x[1]*y[0])+(x[1]*y[1])+(x[1]*y[2])+(x[1]*y[3])+
            (x[1]*y[4])+(x[1]*y[5])+(x[1]*y[6])+(x[1]*y[7]);
        ...
        s7+=(x[7]*y[0])+(x[7]*y[1])+(x[7]*y[2])+(x[7]*y[3])+
            (x[7]*y[4])+(x[7]*y[5])+(x[7]*y[6])+(x[7]*y[7]);
    }
    return s0+s1+s2+s3+s4+s5+s6+s7;
}
Apex-MAP Kernel Routine: Assembler Code for -O2

Intel(R) C++ Compiler for IA-64, Ver. 10.1: icc -S -O2 compute.c

Itanium: 128 floating point registers

- 2 x ld8 basis address of x[] und y[]
- START LOOP
  - 8 x stfd for s0,...,s7
  - dummy() call
  - 8 x ldfd for s0,...,s7 (8 floating point registers)
  - 8 x ldfd for x[0],...,x[7] (8 floating point registers)
  - 8 x ldfd for y[0],...,y[7] (8 floating point registers)
  - 64 x consecutive bundles with 1 fma per bundle
    { .mfi
      nop.m 0
      fma.d f15=f39,f47,f12
      nop.i 0
    }
    e.g. for s0:

    f15=f39*f47+f12 f15=x[0]*y[7]+s0
    f07=f39*f38+f15 f07=x[0]*y[6]+x[0]*y[7]+s0
    f09=f39*f35+f07 f09=x[0]*y[5]+x[0]*y[6]+x[0]*y[7]+s0

- STOP LOOP

- 8 x stfd for s0,...,s7
- 7 x fadd to compute s=s0+s1+s2+s3+s4+s5+s6+s7
  (fadd = fma with f?=f?,f1,f? (pure addition, since f1=1.))
Performance Counters (on Itanium):

- Number of floating point operations per cycle:
  \[(\text{FP\_OPS\_RETIRE}\_D / \text{CPU\_OP\_CYCLES\_ALL})\]

- Memory bandwidth, expressed by L3 misses in Bytes/cycle:
  \[(L3\_MISS \_\text{ALL} \times \text{L3 cacheline size})\]

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Itanium2 Montecito</th>
<th>Nehalem-EP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock rate</td>
<td>1.6 GHz</td>
<td>2.53 GHz</td>
</tr>
<tr>
<td>max. Flops per cycle</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>peak performance per core</td>
<td>6.4 GFlop/s</td>
<td>10.12 GFlop/s</td>
</tr>
<tr>
<td>L3 cache</td>
<td>9 MB</td>
<td>8 MB</td>
</tr>
<tr>
<td>L3 cacheline size</td>
<td>128 Bytes</td>
<td>64 Bytes</td>
</tr>
<tr>
<td>Bandwidth to memory</td>
<td>8.5 GBytes/s</td>
<td>25.6 GB/s</td>
</tr>
<tr>
<td></td>
<td>5.3 Bytes/cycle</td>
<td>10.1 Bytes/cycle</td>
</tr>
</tbody>
</table>
Modelling of HLRB-II Performance using Apex-MAP

Number of floating point operations per cycle ($\frac{FP_{OPS_{\text{RETIRED}}}}{CPU_{OP_{Cycles_{\text{ALL}}}}}$) versus the memory bandwidth, expressed by L3 misses in Bytes/cycle ($\frac{L3_{\text{MISSES}}}{CPU_{OP_{Cycles_{\text{ALL}}}}} \times 128$ Bytes).

- Real applications: Apex-MAP
  - Apex covers 98.9%
  - Around 3 days, sampling every 10 min
  - 3254784 samples

- Various Apex-MAP parameters
  - 23226 samples

Volker Weinberg, LRZ
LRZ - 2.3.2010

The Apex-MAP Benchmark
Modelling of HLRB-II Performance using Apex-MAP

Number of floating point operations per cycle ($FP_{OPS_{\text{RETIRIED}}} / CPU\_OP\_CYCLES\_ALL$) versus the memory bandwidth, expressed by L3 misses in Bytes/cycle ($L3\_MISSES/CPU\_OP\_CYCLES\_ALL \times 128$ Bytes).

real applications

(≈ 3 days, sampling every 10 min)
3254784 samples

Apex-MAP

various Apex-MAP parameters
23226 samples
Modelling of HLRB-II Performance using Apex-MAP

strided access memory pattern

random access memory patterns
Overall mean Apex-MAP performance on HLRB-II: 0.898 GFlop/s per core
Measured HLRB-II performance (3-day interval): 0.48 Flop/cycle \times 1.6 \text{ GHz} = 0.768 \text{ GFlop/s}
Mathematical kernels used for the validation:

- **mod2am**: Dense matrix-matrix multiplication
- **mod2as**: Sparse matrix-vector multiplication

Validating Apex-MAP by using the two mathematical kernels needs several steps:

1. Measure the performance of mod2am/as on the original hardware (HLRB II).
2. Measure the hardware counters for mod2am/as on HLRB II.
3. Generate weights for each square and each kernel.
4. Measure the performance of mod2am/as on the target hardware (Nehalem EP).
5. Run Apex-MAP with the weights for mod2am/as on Nehalem and HLRB II.
6. Compare the predicted results (Step 5) with the actual results (Steps 1 and 4).
Validation of Apex-MAP: Measurements on HLRB-II

The Apex-MAP Benchmark

Volker Weinberg, LRZ

LRZ· 2.3.2010

The Apex-MAP Benchmark
Validation of Apex-MAP: Predictions on HLRB-II

measured mean perf. of mod2am/s:
5.4 Gflop/s  
(84% peak) 

measured mean perf. of mod2as:
0.5 GFlop/s  
(8% peak)
Validation of Apex-MAP: Predictions on Nehalem-EP

measured mean perf. of mod2am/s:

8 Gflop/s
(80% peak)

0.9 GFlop/s
(9% peak)
Apex-MAP Kernel Routine: Perform. on Nehalem-EP

icc -xSSE4.2 -O0
icc -xSSE4.2 -O1
icc -xSSE4.2 -O2
icc -xSSE4.2 -O3

% of Peak

Intel C++ Comp. for Intel 64, Vers. 11.1
SSE2/3 Floating Point Instructions

x86-64: 16 x 128 bit MMX Registers %xmm0, ..., %xmm15 → 32 doubles

SSE2 Packed Double Precision Data Type

SSE2 Vertical Operations addpd /mulpd

SSE3 Horizontal Addition haddpd
Apex-MAP Kernel Routine: Assembler Code for -O2

Intel C++ Comp. for Intel 64, Vers. 11.1: icc -S -O2 -xSSE4.2 compute.c

x86-64: 16 x 128 bit MMX Registers %xmm0, ..., %xmm15 → 32 doubles

- START LOOP
  - dummy() call
  - 4 x movaps fuer y[0],...,y[7]
    movaps y(%rip), %xmm9 %xmm9= y[0] | y[1]
movaps 16+y(%rip), %xmm8 %xmm8= y[2] | y[3]
movaps 32+y(%rip), %xmm7 %xmm7= y[4] | y[5]
movaps 48+y(%rip), %xmm6 %xmm6= y[6] | y[7]

  - fuer jedes s0,...s7
    movsd (%rsp), %xmm6 %xmm6= s0 | 0.
movddup x(%rip), %xmm3 %xmm3= x[0] | x[0]
movaps %xmm3, %xmm5 %xmm5= x[0] | x[0] (not for s7)
movaps %xmm3, %xmm2 %xmm2= x[0] | x[0] (not for s7)
movaps %xmm3, %xmm4 %xmm4= x[0] | x[0] (not for s7)
mulpd %xmm9, %xmm5 %xmm5= x[0]*y[0] | x[0]*y[1]
addpd %xmm5, %xmm6 %xmm6= s0+x[0]*y[0] | x[0]*y[1]
mulpd %xmm8, %xmm2 %xmm2= x[0]*y[2] | x[0]*y[3]
mulpd %xmm7, %xmm4 %xmm4= x[0]*y[4] | x[0]*y[5]
addpd %xmm4, %xmm6 %xmm6= s0+x[0]*y[0]+x[0]*y[2]+x[0]*y[6] | x[0]*y[1]+x[0]*y[3]+x[0]*y[7]
mulpd %xmm0, %xmm3 %xmm3= x[0]*y[6] | x[0]*y[7]
addpd %xmm6, %xmm5 %xmm5= x[0]*y[2]+x[0]*y[6] | x[0]*y[3]+x[0]*y[7]
addpd %xmm2, %xmm3 %xmm3= x[0]*y[2]+x[0]*y[6] | x[0]*y[3]+x[0]*y[7]

--> 32 x addpd (SSE2), 32 x mulpd (SSE2), 8 x haddpd (SSE3), 8 x movsd (SSE2) to load s0,...,s7,
     8 x movddup (SSE3) to load x[i] | x[i] , 7x3 x movaps (SSE) to copy x[i] | x[i]

- STOP LOOP

7 x addsd to compute s=s0+s1+s2+s3+s4+s5+s6+s7

7 x addsd (SSE2)
Apex-MAP benchmark can be used to model our application mix on HLRB-II → Apex-MAP is well suited for benchmarking the suitability of new hardware architectures, e.g. multi- and manycore CPUs, systems with 10000+ processors, systems with accelerators (GPGPUs, CELL processors, ...).

It has to be assured that the 128 Floating point operations in the loop body are really executed and not cancelled by optimisations of the compiler.

Future work: Investigation of the quality of predictions; refine the benchmark (kernel routine) that it adopts easily to new environments.

Vision: Implementing Apex-MAP in a language that supports both multi-core CPUs, GPGPUs and the CELL processor (currently only RapidMind) could offer an easy way to simulate typical application performance patterns on a broad range of architectures.
E. Strohmaier, H. Shan
Architecture Independent Performance Characterization and Benchmarking for Scientific Applications
Volendam, The Netherlands, Oct. 2004
https://ftg.lbl.gov/ApeX/mascots.pdf

R. Patra, M. Brehm, R. Bader, R. Ebner, S. Haupt
Performance Monitoring – A Generic Approach
(LRZ-Bericht 2006-06)
http://www.lrz-muenchen.de/wir/berichte/TB/LRZ-Bericht-2006-06.pdf

Volker Weinberg, Matthias Brehm, Iris Christadler
OMI4papps: Optimisation, Modelling and Implementation for Highly Parallel Applications
Acknowledgements

- Matthias Brehm, Iris Christadler, LRZ, Germany
- KONWIHR-II project OMI4papps: Optimisation, Modelling and Implementation for Highly Parallel Applications
- PRACE project funded in part by the EU’s 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528