Code Performance Analysis
Massimiliano Fatica

ASCI TST Review
May 8 2003
Performance

- Theoretical peak performance of the ASCI machines are in the Teraflops range, but sustained performance with real applications is far from the peak

  - Salinas, one of the 2002 Gordon Bell Awards, was able to sustain 1.16 Tflops on ASCI White (less than 10% of peak)

- On the Earth Simulator, a custom engineered system with exceptional memory bandwidth, interconnect performance and vector processing capabilities

  - Global atmospheric simulation was able to achieve 65% of the 40 Tflops of peak performance
Applications

Our main applications, CDP and TFLO, are coded in F90 and use MPI for message passing.

- **CDP**: LES unstructured finite volume code for the combustor:
  - Uses a lot of advanced features of F90

- **TFLO**: RANS multi-block structured finite volume code for the turbo-machinery.
  - Uses F90 features mainly for flexible data-structures (derived data type and pointer)
Our objective is to have portable, fast and scalable codes.

Portability and performance are often conflicting requirements.

Strike a balance between codes that are easy to maintain and perform well.
To achieve high performance, particular attention needs to be devoted to unrolling, software pipelining, etc.

We express our intent in the code and let the compiler do the tuning:

- Hand tuning is going to affect portability
- The compiler usually does a better job
Example

do i=1,n
   A(i) = A(i) + B(i)*C
end do

<table>
<thead>
<tr>
<th>Simple code</th>
<th>Unrolled 4x</th>
<th>Unrolled 4x, pipelined</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ldt Ai; ldt Bi</td>
<td>1 ldt Bi; ldt Bi+1</td>
<td>1 ldt Bi ; ldt Bi+1; mult Bi-4; addt Ai-8</td>
</tr>
<tr>
<td></td>
<td>2 ldt Bi+2; ldt Bi+3</td>
<td>2 ldt Bi+2 ; ldt Bi+3; mult Bi-3; addt Ai-7</td>
</tr>
<tr>
<td></td>
<td>3 ldt Ai; ldt Ai+1</td>
<td>3 ldt Ai ; ldt Ai+1; mult Bi-2; addt Ai-6</td>
</tr>
<tr>
<td></td>
<td>4 ldt Ai+2; ldt Ai+3</td>
<td>4 ldt Ai+2 ; ldt Ai+3; mult Bi-1; addt Ai-5</td>
</tr>
<tr>
<td></td>
<td>5 mult Bi</td>
<td>5 stt Ai-12; stt Ai-11</td>
</tr>
<tr>
<td></td>
<td>6 mult Bi+1</td>
<td>6 stt Ai-10; stt Ai-9; bne loop</td>
</tr>
<tr>
<td></td>
<td>7 mult Bi+2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 mult Bi+3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 addt Ai</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10 addt Ai+1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11 addt Ai+2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12 addt Ai+3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>13 stt Ai</td>
<td></td>
</tr>
<tr>
<td></td>
<td>14 stt Ai+1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15 stt Ai+2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16 stt Ai+3; bne loop</td>
<td></td>
</tr>
</tbody>
</table>

13 cycles per iteration  4 iteration in 16 cycles = 4 cycles per iteration  4 iteration in 6 cycles = 1.5 cycles

Alpha EV68: 4 cycles for f.p. load from cache; f.p. add and multiply 4 cycles (pipelined)
Performance tuning

- Coding style
- Compilers
- Message passing implementation
Code style / compiler interaction

integer, parameter:: rfp=kind(0.d0)

  type adt_type
    real(kind=rfp), dimension(:,:,:), pointer :: Ex, Ey, Ez, Hx, Hy, Hz
  end type adt_type

real(kind=rfp), dimension(:,:,:), allocatable :: Ex, Ey, Ez, Hx, Hy, Hz

  type(adt_type) :: a

call update_array(Ex,Ey,Ez,Hx,Hy,Hz,n)

call update_adt(a,n)

call update_array(a%Ex,a%Ey,a%Ez,a%Hx,a%Hy,a%Hz,n)
# Effect of coding style

All values in MFlops

<table>
<thead>
<tr>
<th></th>
<th>ASCI Q Compaq F90</th>
<th>Blue Horizon xlf</th>
<th>Origin 300 Mips Pro</th>
<th>P4 IFC 7.1</th>
<th>P4 PGI 4.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using F77 style arrays</td>
<td>1509</td>
<td>543</td>
<td>545</td>
<td>749</td>
<td>423</td>
</tr>
<tr>
<td>Using derived data type</td>
<td>665</td>
<td>395</td>
<td>382</td>
<td>369</td>
<td>47</td>
</tr>
<tr>
<td>Using components of the derived data type</td>
<td>1501</td>
<td>191</td>
<td>532</td>
<td>743</td>
<td>423</td>
</tr>
</tbody>
</table>
The current version of TFLO stores everything in 1D arrays and uses starting indices for every block and multigrid level.

The new version of TFLO uses derived datatype and pointers

- The new version is 10-20% slower
- The new version is more readable and maintainable
- Implementing new algorithms and turbulence models is much easier
Code performance

- Machine: FROST (IBM SP3, Power3 at 375 Mhz) Peak rate 1500 Mflops

- CDP: LES of a reacting flow in a coaxial combustor
  GRID 2.5 million control volumes 64 proc total memory 3GB or 1.22 GB/million CV’s
  Performance measured with hpmcount: 87 MFlops

- TFLO: 210 Mflops
CDP scalability test

![Graph showing CDP scalability test results. The graph plots speedup against the number of processors. The data points for BLUE (Livermore), FROST (Livermore), and RED (Sandia) are shown, with the ideal scalability line indicated. The x-axis represents the number of processors, ranging from 0 to 1200, and the y-axis represents speedup, ranging from 0 to 1200. The graph also notes 16 million control volumes.](image)
TFLO scalability test
Queue systems

The interaction with the queue system is an important factor in the choice of the number of nodes for a run:

- Lower bound: memory needed by the code
- Upper bound: number of CPUs available
- Frequency of high node count availability is very low
Performance metric

★ What is really important, it is the Time to Solution

- I/O can account for a large portion of the runtime
- Improving the pre/post processing steps

★ The real gain usually comes from algorithm improvement
Future work

We are going to devote more efforts to performance analysis once the code implementation is complete:

- improve the single node performance
- improve the network performance