Multicore Challenge in Vector Pascal

P Cockshott, Y Gdura
N-body Problem

• Part 1 (Performance on Intel Nehalem)
  • Introduction (Vector Pascal, Machine specifications, N-body algorithm)
  • Data Structures (1D and 2D layouts)
  • Performance of single thread code (C and Vector Pascal)
  • Performance of multithread code (VP SIMD version)
  • Summary Performance on Nehalem

• Part 2 (Performance on IBM Cell)
  • Introduction
  • New Cell-Vector Pascal (CellVP) Compiler
  • Performance on Cell (C and Vector Pascal)
Vector Pascal

• Extends Pascal’s support for array operations

• Designed to make use of SIMD instruction sets and multi-core
Xeon Specifications

- **Hardware**
  - Year 2010
  - 2 Intel Xeon Nehalem (E5620) - 8 cores
  - 24 GB RAM, 12MB cache
  - 16 threads
  - 2.4 GHz

- **Software**
  - Linux
  - Vector Pascal compiler
  - GCC version 4.1.2
The N body Problem

For 1024 bodies

Each time step

   For each body B in 1024
      Compute force on it from each other body
      From these derive partial acceleration
      Sum the partial accelerations
      Compute new velocity of B

   For each body B in 1024
      Compute new position
Data Structures

The C implementation stores the information as an array of structures each of which is

```
struct planet {
    double x, y, z;
    double vx, vy, vz;
    double mass;
};
```

Does not align well with cache or SIMD registers
Alternative Horizontal Structure

This layout aligns the vectors with the cache lines and with the vector registers
The Reference C Version

```c
for (i = 0; i < nbodies; i++) {
    struct planet * b = &(bodies[i]);
    for (j = i + 1; j < nbodies; j++){
        struct planet * b2 = &(bodies[j]);
        double dx = b->x - b2->x;
        double dy = b->y - b2->y;
        double dz = b->z - b2->z;
        double distance = sqrt(dx * dx + dy * dy + dz * dz);
        double mag = dt / (distance * distance);
        b->vx -= dx * b2->mass * mag;
        b->vy -= dy * b2->mass * mag;
        b->vz -= dz * b2->mass * mag;
        b2->vx += dx * b->mass * mag;
        b2->vy += dy * b->mass * mag;
        b2->vz += dz * b->mass * mag;
    }
}
```

Note that this version has side effects so the successive iterations of the outer loop can not run in parallel as the inner loop updates the velocities.
Equivalent Record Based Pascal

row:=0;
  b := planets[i];
  for j := 1 to n do begin
    b2 := planets[j];
    dx := b^.x - b2^.x;
    dy := b^.y - b2^.y;
    dz := b^.z - b2^.z;
    distance := sqrt(dx * dx + dy * dy + dz * dz);
    mag := dt*b2^.mass / (distance * distance * distance+epsilon);
    row[1] :=row[1]- dx * mag;
  end;

This is side effect free as the total change in the velocity of the ith planet is built up in a local row vector which is added to the planet velocities later.
Complexity and Performance Comparison

Timings below are for single threaded code on Xeon

<table>
<thead>
<tr>
<th></th>
<th>Vector Pascal</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unoptimised</td>
<td>28.9 ms</td>
<td>30 ms</td>
</tr>
<tr>
<td>-O3</td>
<td>23.5 ms</td>
<td>14 ms</td>
</tr>
</tbody>
</table>

Note: Pascal performs $N^2$ operations while C does $N^2/2$
SIMD friendly version – no explicit inner loop

pure function computevelocitychange(start:integer):coord;

-- declarations {M: pointer to mass vector, x: pointer to position matrix, di :
displacement matrix, distance: vector of distances}

begin

row:=x^[iota[0],i];
{ Compute the displacement vector between each planet and planet i.}
di:= row[iota[0]]- x^;
{ Next compute the euclidean distances }
xp:=@ di[1,1]; yp:=@di[2,1];zp:=@di[3,1]; { point at the rows }
distance:= sqrt(xp^*xp^+ yp^*yp^+ zp^*zp^)+epsilon;
mag:=dt/(distance *distance*distance );
changes.pos:= \+ (M^*mag*di);

end

Row Summation operator builds x,y,z components of dv
Pack this up in Pure Function Applied in Parallel

```pascal
procedure radvance(    dt:real);
var dv:array[1..n,1..1] of coord; i,j:integer;
pure function computevelocitychange(i:integer;dt:real):coord;
begin
{--- do the computation on last slide}
computevelocitychange:=changes.pos;
end;
begin
  dv := computevelocitychange(iota[0],dt); { can be evaluated in parallel}
  for i:= 1 to N do 
    for j:= 1 to 3 do
      v^[j,i] := v^[j,i] + dv[i,1].pos[j]; { update velocities }
      x^ := x^ + v^ * dt; 
    { Finally update positions. }
end;
```
Now Compile with the Multiple Cores

- Programme unchanged compiled with from 1 to 16 cores for example
- `vpc V12 –cpugnuP4 –cores8`
- X axis threads, Y axis time in seconds, log log plot, 256 runs
- Mean time for 7 cores = 5.2 ms
Combined SIMD Multicore Performance

<table>
<thead>
<tr>
<th>Type</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>V12 record</td>
<td>$y = 0.0229x^{-0.895}$</td>
</tr>
<tr>
<td>v8simdhyper</td>
<td>$y = 0.009x^{-0.448}$</td>
</tr>
<tr>
<td>non hyper</td>
<td>$y = 0.0135x^{-0.842}$</td>
</tr>
<tr>
<td>C version</td>
<td></td>
</tr>
<tr>
<td>V12 rec hyper</td>
<td>$y = 0.0284x^{-0.768}$</td>
</tr>
</tbody>
</table>

- Power (V12 record)
- Power (v8simdhyper)
- Power (non hyper)
- Power (C version)
- Power (V12 rec hyper)
Summary Time per Iteration

Best performance on the Xeon was using 7 cores:

- SIMD performance scales as $c^{0.84}$
- Record performance scales as $c^{0.89}$, where $c$ the number of cores.

<table>
<thead>
<tr>
<th></th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C optimised 1 core</td>
<td>14 ms</td>
</tr>
<tr>
<td>SIMD code Pascal 1 core</td>
<td>16 ms</td>
</tr>
<tr>
<td>SIMD code Pascal 7 cores</td>
<td>0.25 ms</td>
</tr>
<tr>
<td>Record code Pascal 1 core</td>
<td>23 ms</td>
</tr>
<tr>
<td>Record code Pascal 7 cores</td>
<td>0.75 ms</td>
</tr>
</tbody>
</table>
Performance in GFLOPS

- We pick the 6 core versions as it gives the peak flops, being just before the hyper-threading transition.
- This transition affects 7 and 8 thread versions.

<table>
<thead>
<tr>
<th>Op.’s per Body</th>
<th>Vector Pascal</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute displacement</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>get distance</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>compute mag</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>evaluate dv</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>total per inner loop</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>times round inner loop</td>
<td>1024</td>
<td>512</td>
</tr>
<tr>
<td>times round outer loop</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>total per timestep</td>
<td>20971520</td>
<td>15728640</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language / version</th>
<th>Time (msec)</th>
<th>Number Of Cores</th>
<th>GFLOPS Total</th>
<th>Per Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD version Pascal</td>
<td>14.36</td>
<td>1</td>
<td>1.460</td>
<td>1.460</td>
</tr>
<tr>
<td>SIMD version Pascal</td>
<td>2.80</td>
<td>6</td>
<td>7.490</td>
<td>1.248</td>
</tr>
<tr>
<td>record version Pascal</td>
<td>23.50</td>
<td>1</td>
<td>0.892</td>
<td>0.892</td>
</tr>
<tr>
<td>record version Pascal</td>
<td>4.23</td>
<td>6</td>
<td>4.958</td>
<td>0.826</td>
</tr>
<tr>
<td>C version</td>
<td>14.00</td>
<td>1</td>
<td>1.123</td>
<td>1.123</td>
</tr>
</tbody>
</table>
N-Body on Cell

- The Cell Architecture
- The CellVP Compiler using Virtual SIMD Machine
- Alignment and Synchronization
- Performance on Cell
The Cell Heterogeneous Architecture

• Year 2007

• Processors
  • 1 PowerPC (PPE) , 3.2 GHz, 512 MB RAM, 512KB L2, 32KB L1 cache
  • 8 Synergistic processors (SPEs), 3.2 GHz, 256 KB

• 2 Different Instruction sets ( 2 Different Compilers)

• Memory Flow Controller (MFC) on each SPE. (DMA, Mailbox, signals )

• Alignment boundary (16 bytes or 128bytes for better performance)

• Existing Supported Languages (C/C++ and Fortran)
The CellVP Compiler System

• **Objective**
  Automatic parallelizing compiler using virtual machine model

• **Aim at**
  Array expressions in intensive-data applications.

• **Built of**
  1. A PowerPC compiler
  2. A Virtual SIMD Machine (VSM) model to access the SPEs.
• The PowerPC Compiler

• Transform sequential VP code into PPE code

• Convert large array expression into VM instructions

• Append to the prologue code, code to launch threads on the SPEs

• Append to the epilogue code, code to terminate SPEs’ threads.
Virtual SIMD Machine (VSM) Model

• VSM Instructions

• Register to Register Instructions

• Operate on virtual SIMD registers (1KB - 16KB)

• Support basic Operations (+, - , / , * , sqrt , \+, rep ... etc)
VSM Interpreter

1. The PPE Opcode dispatcher

i. Chops data equally on used SPEs
ii. Formatting messages (opcode, registers to be used, starting address)
iii. Writing messages to SPEs’ Inbound mailbox
iv. Waiting for a completion acknowledgment from SPEs (blocking mode)

2. The SPE Interpreter (A program runs in a background)

i. Checks Inbound mailbox for new messages
ii. On receiving a message, an SPE performs the required operation
iii. Sends an acknowledgment with the completion (If needed)
The CellVP Compiler System

1. Generates PowerPC machine instructions (sequential code)

2. Generates VSM instructions to evaluate large arrays on SPEs.

3. PPE Handles
   1. Data Partitioning on SPEs
   2. Communication (Mailboxes)

4. SPE Handles
   1. Alignment (load & Store)
   2. Synchronization

   Parts of data that might being processed on the preceding SPE and succeeding SPE
Virtual SIMD Register Chopped on 4 SPEs

Alignment & Synchronization (Store Operation)

Virtual Register 4KB

Block0-SPE0  Block1-SPE1  Block2-SPE2  Block3-SPE3

Actual Starting Address

Aligned address

3 DMA Transfers

Data Block Size (1KB)

1st DMA SPE1  2nd DMA SPE1  1st DMA SPE2
Sets lock on 128B

1st DMA SPE3
Sets lock on 128B

Virtual SIMD Register Chopped on 4 SPEs
N-Body Problem on the Cell

Code: Same Xeon version

Data Structure: large scale (4KB) Horizontal Structure

Machine: PS3 (only four SPEs used)

Compilers: GNU C/C++ compiler version 4.1.2
Vector Pascal “CellVP”
Performance of VP&C on Xeon & Cell (GFLOPS)

<table>
<thead>
<tr>
<th>Op.’s per Body</th>
<th>Vector Pascal</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute displacement</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>get distance</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>compute mag</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>evaluate dv</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>total per inner loop</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>times round inner loop</td>
<td>1024</td>
<td>512</td>
</tr>
<tr>
<td>times round outer loop</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>total per time step</td>
<td>20971520</td>
<td>15728640</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language / version</th>
<th>Time (msec)</th>
<th>Number of Cores</th>
<th>Total GFLOPS</th>
<th>Per Core GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD version Pascal</td>
<td>14.36</td>
<td>1</td>
<td>1.460</td>
<td>1.460</td>
</tr>
<tr>
<td>SIMD version Pascal</td>
<td>2.80</td>
<td>6</td>
<td>7.490</td>
<td>1.248</td>
</tr>
<tr>
<td>record version Pascal</td>
<td>23.50</td>
<td>1</td>
<td>0.892</td>
<td>0.892</td>
</tr>
<tr>
<td>record version Pascal</td>
<td>4.23</td>
<td>6</td>
<td>4.958</td>
<td>0.826</td>
</tr>
<tr>
<td>C version</td>
<td>14.00</td>
<td>1</td>
<td>1.123</td>
<td>1.123</td>
</tr>
<tr>
<td>Cell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pascal (PPE)</td>
<td>381</td>
<td>1</td>
<td>0.055</td>
<td>0.055</td>
</tr>
<tr>
<td>Pascal (SPE)</td>
<td>105</td>
<td>1</td>
<td>0.119</td>
<td>0.119</td>
</tr>
<tr>
<td>Pascal (SPEs)</td>
<td>48</td>
<td>4</td>
<td>0.436</td>
<td>0.109</td>
</tr>
<tr>
<td>C (PPE, O3)</td>
<td>45</td>
<td>1</td>
<td>0.349</td>
<td>0.349</td>
</tr>
</tbody>
</table>
## VP Performance on Large Problems

<table>
<thead>
<tr>
<th>N-body Problem Size</th>
<th>Performance (seconds) per Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vector Pascal</td>
</tr>
<tr>
<td></td>
<td>PPE</td>
</tr>
<tr>
<td>1K</td>
<td>0.381</td>
</tr>
<tr>
<td>4K</td>
<td>4.852</td>
</tr>
<tr>
<td>8K</td>
<td>20.355</td>
</tr>
<tr>
<td>16K</td>
<td>100.250</td>
</tr>
</tbody>
</table>
Log log chart of performance of the Cell

\[ y = 20.5x^{-0.813} \]

\[ y = 97.9x^{-0.9} \]

Time in (secs)

Degree of FPU parallelism

8k
16K
Power (8k)
Power (16K)
THANK YOU ANY?