Performance Tuning of Ateles using Xevolver

Kazuhiko Komatsu
Tohoku University
11 October, 2017
WSSP26
Introduction

• Increase of variety of HPC systems
  – Scalar-type system
    • A large number of scalar processors with many cores and large cache
    • Massively parallel calculations
  – Accelerator-type system
    • Accelerators with lots of simple cores
    • Data parallel calculations
  – Vector-type system
    • Vector cores with a high memory bandwidth
    • calculates set of data elements at one time
  – TOP500-specialized system

System-dependent information has to be written in a code to exploit the potential of each HPC system
Problems

• By effectively utilizing the features of each HPC system
  – Directly writing system-dependent information
    • C, Fortran, CUDA/OpenCL, AVX intrinsic, ...
    • **Several versions** of an application code (or IF/DEF HELL)
      – Each version is optimized for an HPC system
    • **Low readability and maintainability**
      – System-dependent optimizations may be obstacle of application behavior
Objective and Approach

• Objective
  – System-dependent optimizations while keeping readability and maintainability

• Approach
  – Utilize code transformations for system-dependent optimizations
    • Xevolver: Code transformation framework
    • Xevtgen: Fortran-like rule generator
      – The original code as unchanged as possible
Case study: Tuning of Ateles for SX-ACE

• Overview of Ateles
  – One of solvers in the APES simulation suite
    • Fortran 2008/2003/95/90, python, OpenMP/MPI
      – ISO C interface
      – function pointer
      – allocatable variables in structures
  – Discontinuous Galerkin Solver
    • Fortran 100K lines
    • Python 3.5K lines
Preliminary Evaluation on SX-ACE

• Test data
  – testsuite/sx_testing/ateles.lua

• Parameter survey in the script

```lua
-- Parameters to vary --
--
-- The polynomial degree specifies the accuracy within each element and defines
-- the number of degrees of freedom per element that will be (degree+1)^3 for
-- a poly space of Q
degree = 7

-- ...the uniform refinement level for the periodic cube
level = 2

-- "level" does not affect the performance
-- "degree" might affect the vector length
```
# Single-node performance

<table>
<thead>
<tr>
<th></th>
<th>degree = 7</th>
<th>level = 2</th>
<th>degree = 14</th>
<th>level = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Time (sec)</td>
<td>83.176829</td>
<td>811.614073</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Time (sec)</td>
<td>82.835361</td>
<td>811.393898</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sys Time (sec)</td>
<td>0.043444</td>
<td>0.062352</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector Time (sec)</td>
<td>12.100865</td>
<td>134.764311</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inst. Count</td>
<td>25562226844</td>
<td>542853655029</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V. Inst. Count</td>
<td>799528202</td>
<td>8621373999</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V. Element Count</td>
<td>61705955881</td>
<td>1262989794226</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V. Load Element Count</td>
<td>7714451369</td>
<td>126354804785</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOP Count</td>
<td>9218093212</td>
<td>207136982518</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOPS</td>
<td>1043.861625</td>
<td>2214.981009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MFLOPS</td>
<td>111.282103</td>
<td>255.285359</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A. V. Length</td>
<td>77.17796</td>
<td>146.495187</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V. Op. Ratio (%)</td>
<td>71.362225</td>
<td>70.274554</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Size (MB)</td>
<td>256.03125</td>
<td>320.03125</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIPS</td>
<td>308.590758</td>
<td>669.038375</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I-Cache (sec)</td>
<td>0.266876</td>
<td>1.291097</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O-Cache (sec)</td>
<td>56.223546</td>
<td>377.723006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bank Conflict Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU Port Conf. (sec)</td>
<td>0.331255</td>
<td>8.065057</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Network Conf. (sec)</td>
<td>3.205328</td>
<td>45.384267</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADB Hit Element Ratio (%)</td>
<td>62.779898</td>
<td>51.393439</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Detailed cost distribution

**degree = 7 | level = 2**

<table>
<thead>
<tr>
<th>PROC. NAME</th>
<th>FREQ.</th>
<th>EXCL. TIME[sec]</th>
<th>%</th>
<th>AVER.TIME [msec]</th>
<th>MFLOPS</th>
<th>V.OP RATIO</th>
<th>AVER. V.LEN</th>
<th>VECTOR TIME</th>
<th>BANK CONFLICT</th>
<th>ADB HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODG_VOLTOFACE_Q</td>
<td>4584</td>
<td>69.65</td>
<td>83.8</td>
<td>15.194</td>
<td>25.9</td>
<td>53.85</td>
<td>151.1</td>
<td>1.572</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>MODG_PRJ_PFLUX3_Q_6</td>
<td>48896</td>
<td>3.009</td>
<td>3.6</td>
<td>0.062</td>
<td>297.7</td>
<td>98.96</td>
<td>38.5</td>
<td>2.986</td>
<td>0.05</td>
<td>1.671</td>
</tr>
<tr>
<td>MAXWELL_FLUX_CUBE_VEC</td>
<td>2292</td>
<td>2.403</td>
<td>2.9</td>
<td>1.048</td>
<td>214.9</td>
<td>80.25</td>
<td>39.1</td>
<td>2.181</td>
<td>0.009</td>
<td>0.043</td>
</tr>
<tr>
<td>MODG_PRJ_PFLUX2_Q_6</td>
<td>48896</td>
<td>1.594</td>
<td>1.9</td>
<td>0.033</td>
<td>562</td>
<td>98.86</td>
<td>38.6</td>
<td>1.563</td>
<td>0.061</td>
<td>0.254</td>
</tr>
<tr>
<td>MODG_PRJ_PFLUX1_Q_6</td>
<td>48896</td>
<td>1.518</td>
<td>1.8</td>
<td>0.031</td>
<td>589.8</td>
<td>98.87</td>
<td>38.8</td>
<td>1.476</td>
<td>0.03</td>
<td>0.165</td>
</tr>
</tbody>
</table>

**degree = 14 | level = 2**

<table>
<thead>
<tr>
<th>PROC. NAME</th>
<th>FREQ.</th>
<th>EXCL. TIME[sec]</th>
<th>%</th>
<th>AVER.TIME [msec]</th>
<th>MFLOPS</th>
<th>V.OP RATIO</th>
<th>AVER. V.LEN</th>
<th>VECTOR TIME</th>
<th>BANK CONFLICT</th>
<th>ADB HIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODG_VOLTOFACE_Q</td>
<td>16080</td>
<td>665.772</td>
<td>82.2</td>
<td>41.404</td>
<td>62.6</td>
<td>48.56</td>
<td>245.9</td>
<td>16.892</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>MODG_MAXWELL_PHYSFLUX_CONST</td>
<td>514560</td>
<td>23.831</td>
<td>2.9</td>
<td>0.046</td>
<td>0</td>
<td>95.6</td>
<td>238.3</td>
<td>23.027</td>
<td>3.559</td>
<td>17.651</td>
</tr>
<tr>
<td>MODG_PRJ_PFLUX3_Q_6</td>
<td>171520</td>
<td>23.159</td>
<td>2.9</td>
<td>0.135</td>
<td>1080.5</td>
<td>99.74</td>
<td>130.1</td>
<td>23.075</td>
<td>1.499</td>
<td>8.046</td>
</tr>
<tr>
<td>MODG_PRJ_PFLUX1_Q_6</td>
<td>48896</td>
<td>1.594</td>
<td>1.9</td>
<td>0.033</td>
<td>562</td>
<td>98.86</td>
<td>38.6</td>
<td>1.563</td>
<td>0.061</td>
<td>0.254</td>
</tr>
<tr>
<td>MODG_PRJ_PFLUX2_Q_6</td>
<td>48896</td>
<td>1.518</td>
<td>1.8</td>
<td>0.031</td>
<td>589.8</td>
<td>98.87</td>
<td>38.8</td>
<td>1.476</td>
<td>0.03</td>
<td>0.165</td>
</tr>
</tbody>
</table>

- Cost distributions in both degrees are similar
- Low vector operation ratio in the highest routine

11 October, 2017
WSSP26
Analysis of the routine

MODG_VOLTOFACE_Q
:
2929: +-----+ do iAnsX = 1, maxPolyDegree+1
2930: |       | get the face value of the ansatz function with fixed coordinate
2931: |       | !!faceVal = faceValRightBndAns(iAnsX)
2932: |
2933: |+-----+ do iVar = 1, nScalars
2934: ||+-----+ do iElem=1, nElems
2935: |||+-----+ do facepos = 1, mpd1_square
2936: |||||+-----+ ! get position of the current ansatz function
2937: ||||||+-----+ pos = (facepos-1)*mpd1 + iAnsX
2938: |||||||+-----+ faceState(iElem,facePos,iVar,leftOrRight) &
2939: ||||||||+-----+ & = faceState(iElem,facePos,iVar,leftOrRight) &
2940: |||||||||+-----+ & + volState(iElem,pos,iVar)
2941: ||||||||||+-----+ end do
d2942: |||||||||||+-----+ end do
d2943: ||||||||||++-----+ end do
d2944: ||||||||+-----+ end do
d2945: |||+-----+ end do
d2946: |
d2947: +-----+ end do

• The innermost loop is vectorized, but PARTIALLY
  – Data dependency cannot be solved by the compiler
• The length of the vectorized loop is SHORT

• Compile message
  2936: vec( 2): Partially vectorized loop.
  2936: vec( 24): Iteration count is assumed. Iteration count=5000.
  2936: vec( 25): Work vectors are used. Size=40000byte.
Optimizations effective for SX-ACE

MODG_VOLTOFACE_Q :

2930: +------ do iAnsX = 1, maxPolyDegree+1
2931: |   ! get the face value of the ansatz function with fixed coordinate
2934: | +------ do iVar = 1,nScalars
2938: |   m=nElems
2939: |   n=mpd1_square
2940: |   mn=m*n
2941: |   !cdir nodep
2942: | V------ do iJ= 0, mn - 1
2943: |   iElem=ij / n + 1
2944: |   facepos=mod(ij, n) + 1
2945: |   ! get position of the current ansatz function
2946: |   pos = (facepos - 1)*mpd1 + iAnsX
2947: | V------ end do
2948: +------ end do

Loop Length= ~7
Loop Length= ~6
Loop Length= ~64 * 64

• Compile message
2942: vec( 1): Vectorized loop.
2947: vec( 23): "NODEP" is specified. No dependency is assumed.: FACESTATE
2942: vec( 29): ADB is used for array.: __REAL(KIND=8)

• The length of the vectorized loop increases
  – By collapsing both i loop and j loop
• Successfully vectorized by the “nodep” directive
• Applied for all the same loop nests in the routine
Motivation

• System-aware optimization for SX-ACE
  – Loop collapse and compiler-specific directives
    • Degrade readability
  – Similar optimizations need to be REPEATEDLY applied
    • Loop collapse and insertion of the directive is applied all loop nests in whole code
    • Might lead human errors

The repetitive optimization can be replaced with transformation rules and code transformations
**Xevolver**: code transformation framework

- **Xevolver** [Takizawa, 2014]
  - Can separate system-awareness from a code
    - External custom translation rules and custom directives
  ⇒ Minimize modifications and then keep maintainability of the original code

![Diagram of code transformation process]

Original code → **src2xml** → XML AST → User-defined transformation → XML AST → **xml2src** → Transformed code

Translation rule
**Xevtgen:** Transformation Rule Generator

- **Xevtgen [Suda, 2015]**
  - Easily generation of transformation rules for Xevolver
    - *Dummy Fortran code*
      - Fortran-like code with some special tgen directives
    - *source* and *destination patterns* of a dummy Fortran code
      - Source pattern
        
        ```fortran
        !$xev tgen src begin
        IF (I .EQ. 0) EXIT
        !$xev tgen src end
        ```
      - Destination pattern
        
        ```fortran
        !$xev tgen dst begin
        IF (I == 0) THEN
        EXIT
        END IF
        !$xev tgen dst end
        ```

Standard Fortran programmers can easily learn and generate rules.
Optimization using Xevolver

– Instead of directly modifying a code, custom rules and custom directives are defined using Xevolver

Minimize modifications and then keep maintainability

Original code

XML AST

User-defined transformation

XML AST

Transformed code

Optimized Ateles

Ateles

src2xml

Making rules of modifications

Translation rule

Apply code modification
Loop collapse and NODEP directive

Original

```c
do iVar = 1, nScalars
    do iElem = 1, nElems
        do facepos = 1, mpd1_square
            ...
        end do
    end do
end do
```

```c
m = nElems
n = mpd1_square
mn = m * n

!cdir nodep
```
```
```c
do ij = 0, mn - 1
    iElem = ij / n + 1
    facepos = mod(ij, n) + 1
    ...
end do
```
```

• Code Transformation by Xevolver
  – Just inserting “!$xev collapse” into the original code
  • Can keep the maintainability of the original code as much as possible

11 October, 2017
WSSP26
15
# Transformation rule

<table>
<thead>
<tr>
<th>Source pattern</th>
<th>Destination pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>!$xev tgen src begin</td>
<td>!$xev tgen dst begin</td>
</tr>
<tr>
<td>!$xev collapse</td>
<td>m=nElems</td>
</tr>
<tr>
<td>do iElem=1,nElems</td>
<td>n=mpd1_square</td>
</tr>
<tr>
<td>do facepos = 1,mpd1_square</td>
<td>mn=m*n</td>
</tr>
<tr>
<td>!$xev tgen stmt(body0)</td>
<td>!cdir nodep</td>
</tr>
<tr>
<td>end do</td>
<td>do ij= 0, mn - 1</td>
</tr>
<tr>
<td>end do</td>
<td>iElem=ij / n + 1</td>
</tr>
<tr>
<td>!$xev tgen src end</td>
<td>facepos=mod(ij, n) + 1</td>
</tr>
<tr>
<td></td>
<td>!$xev tgen stmt(body0)</td>
</tr>
<tr>
<td></td>
<td>end do</td>
</tr>
<tr>
<td></td>
<td>!$xev tgen dst end</td>
</tr>
</tbody>
</table>

18 system-aware optimizations can be applied by code transformations
Experimental environment

• Comparison of performance of Ateles on various platforms
  – Original version
  – Tuned version for NEC SX-ACE
    • Loop collapse
    • Insertion of compiler-specific directives

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Xeon E5-2695v2</th>
<th>NEC SX-ACE</th>
<th>IBM Power8</th>
</tr>
</thead>
<tbody>
<tr>
<td>#. of cores</td>
<td>2x 12 cores</td>
<td>4 cores</td>
<td>2x 8 cores</td>
</tr>
<tr>
<td>Mem. Capacity</td>
<td>128 GB</td>
<td>64 GB</td>
<td>512 GB</td>
</tr>
<tr>
<td>Compiler</td>
<td>Intel compiler 16.03</td>
<td>NEC SX 2003 Rev. 061</td>
<td>PGI 16.10 (openpower)</td>
</tr>
</tbody>
</table>
Results

- SX-ACE: x7.5 speedup to original, 18% to Intel original
- Xeon: Tuned version degrades 14%
- IBM: Tuned version accelerates 6%

Optimization for particular platform is not always effective. Our approach that uses transformation is suitable!
Detailed analysis of loop collapse

• degree=7, level=2

- Vector operation ratios become more than 99%
- Vector average lengths become more than 250
Conclusions

• Performance analysis and optimizations of Ateles on SX-ACE
  – Inline expansion
  – Loop collapse
  ➔ Degrades the readability and maintainability
• For high maintainability and performance
  – Code transformation by Xevolver is employed
    • only small number of directives need to be inserted into the original code
• Future work
  – Increase of compatibility of Xevolver
    • The backend of compiler infrastructure does not fully support Fortran2003/2008
      • Need to modify Ateles to avoid the unsupported description