Auto-optimization on parallel hydrodynamic codes: an example of COHERENS with OpenMP for multicore

Francisco López-Castejón, Domingo Giménez

Polytechnic University of Cartagena (Spain), University of Murcia (Spain)
Ecosystem Group, Computational Science Group

June 24, 2010
Guidelines

1. Introduction
   - Objectives
   - About COHERENS
   - Cost Analysis

2. Parallelization
   - Example of parallel code
   - Parallelization experiment
   - Parallelization results

3. Auto-optimization
   - Auto-optimization strategies
   - Results

4. Conclusions
Main Objective

Establish some simple strategies for the development of a parallel code with auto-optimization capacity for multicore system

- Carry out an analysis on a sequential code, COHERENS.
- Extract some loops of the code and parallelize them.
- Analyze the execution time obtained for these loops run in different system and size problems.
- Expose some auto-optimization strategies and the result obtained when we applied it to the loops extracted.
Main Objective

Establish some simple strategies for the development of a parallel code with auto-optimization capacity for multicore system

- Carry out an analysis on a sequential code, COHERENS.
- Extract some loops of the code and parallelize them.
- Analyze the execution time obtained for these loops run in different system and size problems.
- Expose some auto-optimization strategies and the result obtained when we applied it to the loops extracted.
Main Objective

Establish some simple strategies for the development of a parallel code with auto-optimization capacity for multicore system.

- Carry out an analysis on a sequential code, COHERENS.
- Extract some loops of the code and parallelize them.
- Analyze the execution time obtained for these loops run in different system and size problems.
- Expose some auto-optimization strategies and the result obtained when we applied it to the loops extracted.
Main Objective

Establish some simple strategies for the development of a parallel code with auto-optimization capacity for multicore system

- Carry out an analysis on a sequential code, COHERENS.
- Extract some loops of the code and parallelize them.
- Analyze the execution time obtained for these loops run in different system and size problems.
- Expose some auto-optimization strategies and the result obtained when we applied it to the loops extracted.
What’s COHERENS (COupled Hydorodynamical-Ecological model for RegioNal and Shelf seas)

The 3D hydrodynamic model called COHERENS was developed between 1990 and 1998 by the Management Unit of the North Sea Mathematical Models, Napier University, Proudman Oceanographic Laboratory and British Oceanographic Data Center, within the European project MAST PROFILE, NOMADS AND COHERENS.

- It has been used to carry out coastal hydrodynamic studies, not only by Universities but also by private companies.
- It’s opensource software, so we can change the code when we like.
- It has a very structure code, which allow us a greater understanding of how it’s work.
What’s COHERENS (COupled Hydrodynamical-Ecological model for RegioNal and Shelf seas)

The 3D hydrodynamic model called COHERENS was developed between 1990 and 1998 by the Management Unit of the North Sea Mathematical Models, Napier University, Proudman Oceanographic Laboratory and British Oceanographic Data Center, within the European project MAST PROFILE, NOMADS AND COHERENS.

- It has been used to carry out coastal hydrodynamic studies, not only by Universities but also by private companies.
- It’s opensource software, so we can change the code when we like.
- It has a very structure code, which allow us a greater understanding of how it’s work.
What’s COHERENS (COupled Hydrodynamical-Ecological model for RegioNal and Shelf seas)

The 3D hydrodynamic model called COHERENS was developed between 1990 and 1998 by the Management Unit of the North Sea Mathematical Models, Napier University, Proudman Oceanographic Laboratory and British Oceanographic Data Center, within the European project MAST PROFILE, NOMADS AND COHERENS.

- It has been used to carry out coastal hydrodynamic studies, not only by Universities but also by private companies.
- It’s opensource software, so we can change the code when we like.
- It has a very structure code, which allow us a greater understanding of how it’s work.
What’s COHERENS (COupled Hydrodynamical-Ecological model for Regional and Shelf seas)

The 3D hydrodynamic model called COHERENS was developed between 1990 and 1998 by the Management Unit of the North Sea Mathematical Models, Napier University, Proudman Oceanographic Laboratory and British Oceanographic Data Center, within the European project MAST PROFILE, NOMADS AND COHERENS.

- It’s written in Fortran, so we can use free compilers for OpenMP.
- It’s well documented, it’s facilities the job of knowing the function of each routine.
- Having the experience in using this code in carrying out hydrodynamics studies.
What’s COHERENS (COupled Hydrodynamical-Ecological model for Regional and Shelf seas)

The 3D hydrodynamic model called COHERENS was developed between 1990 and 1998 by the Management Unit of the North Sea Mathematical Models, Napier University, Proudman Oceanographic Laboratory and British Oceanographic Data Center, within the European project MAST PROFILE, NOMADS AND COHERENS.

- It’s written in fortran, so we can use free compilers for OpenMP.
- It’s well documented, it’s facilities the job of knowing the function of each routine.
- Having the experience in using this code in carrying out hydrodynamics studies.
What’s COHERENS (COupled Hydrodynamical-Ecological model for RegioNal and Shelf seas)

The 3D hydrodynamic model called COHERENS was developed between 1990 and 1998 by the Management Unit of the North Sea Mathematical Models, Napier University, Proudman Oceanographic Laboratory and British Oceanographic Data Center, within the European project MAST PROFILE, NOMADS AND COHERENS.

- It’s written in fortran, so we can use free compilers for OpenMP.
- It’s well documented, it’s facilities the job of knowing the function of each routine.
- Having the experience in using this code in carrying out hydrodynamics studies.
Loop from subroutine HAD2DV  cost (8 flops) = 7 \ xy + \ y

\[
\begin{align*}
\text{do } i=2,nc \\
\text{do } j=1,nr \\
\text{if } (\text{npi}(j,i).eq.1) \text{ then} \\
\quad \text{ud2}(j,i) = \text{dheeddyvu}(j,i) * ((v2t(c)(j,i)/h2t(c)(j,i) \\
\quad \quad - v2t(c)(j,i-1)/h2t(c)(j,i-1))/g2u(j,i) \\
\quad \quad + \text{sphcur}(j) * \text{ud2}(j,i)/h2t(u(c)(j,i)) \\
\text{endif} \\
\text{end do} \\
\text{end do}
\end{align*}
\]

Cost of the function CRRNT2 = 350 \ xy + 86 \ x + 86 \ y

Francisco López-Castejón, Domingo Giménez

Autooptimization strategies on parallel codes
Number of flops = 350xy + 86x +86y

x = Number of nodes in X axis.
y = Number of nodes in Y axis.
z = Number of levels in Z axis.
Why OpenMP?

- The parallelization in shared memory is the easier to apply in codes with a great number of loops and in multicore systems.
- Multicore systems are widely used: they are on nodes of supercomputers and clusters, and also on laptops and desktops.
- OpenMP is simple to implement in the code.
The parallelization in shared memory is the easier to apply in codes with a great number of loops and in multicore systems. Multicore systems are widely used: they are on nodes of supercomputers and clusters, and also on laptops and desktops. OpenMP is simple to implement in the code.
Why OpenMP?

- The parallelization in shared memory is the easier to apply in codes with a great number of loops and in multicore systems.
- Multicore systems are widely used: they are on nodes of supercomputers and clusters, and also on laptops and desktops.
- OpenMP is simple to implement in the code.
Example of parallel code

```c
!  ! Parallelization
!  ! Auto-optimization

Francisco López-Castejón, Domingo Giménez

Auto-optimization strategies on parallel codes

```
Example of paralell code

c$omp parallel
    c$omp& private (i,j,ydifv,xdifv,ydifu)
c$omp do
        do i=1,nc
            do j=2,nr
                if (npiy(j,i).eq.1) then
                    ydifv = (ydiflv(j,i)-ydiflv(j-1,i))/(gy2v(j)*cosphiv(j))
                    xdifv = 0.5*(xdiflv(j,i+1) + xdiflv(j-1,i+1) - 
                                    xdiflv(j,i) - xdiflv(j-1,i))/gx2v(j,i)
                endif
            endif
            if (i.eq.1) then
                ydifu = (ydiflu(j,i+1) - ydiflu(j,i))
                                    /(0.5*gx2v(j,i+1)+1.5*gx2v(j,i))
            elseif (i.eq.nc) then
                ydifu = (ydiflu(j,i) - ydiflu(j,i-1))
                                    /(0.5*gx2v(j,i-1)+1.5*gx2v(j,i))
            else
                ydifu = (ydiflu(j,i+1) - ydiflu(j,i-1))
                                    /(0.5*(gx2v(j,i-1)+gx2v(j,i)+gx2v(j,i)))
            endif
            vdh2d(j,i) = ydifv + xdifv + ydifu
            vdh2d(j,i) = vdh2d(j,i) + sphcurv(j)*
            (0.5*(xdiflu(j-1,i)+xdiflu(j,i))
                                        -2.0*sphcurv(j)*dheddyvv(j,i)*vd2(j,i)/h2atv(j,i))
        enddo
    enddo

c$omp end do

c$omp end parallel
Parallellizated three types of loop with diferents flops number, 3f, 8f and 19f, and execute in 4 systems:

- Rosebud: 4 itanium dual core. 8 cores. Polytechnic University of Valencia.
- Hipatia: 2 Xeon E5462 quad-core. 8 cores. Polytechnic University of Cartagena.
- Ben: HP Integrity Superdome. 128 cores. Supercomputing Center of the Fundación Parque Científico of Murcia.
- Arabi: 2 Xeon L5450 quad-core. 8 cores. Supercomputing Center of the Fundación Parque Científico of Murcia.
Parallellized three types of loop with different flops number, 3f, 8f and 19f, and execute in 4 systems:

- Rosebud: 4 Itanium dual core. 8 cores. Polytechnic University of Valencia.
- Hipatia: 2 Xeon E5462 quad-core. 8 cores. Polytechnic University of Cartagena.
- Ben: HP Integrity Superdome. 128 cores. Supercomputing Center of the Fundación Parque Científico of Murcia
- Arabi: 2 Xeon L5450 quad-core. 8 cores. Supercomputing Center of the Fundación Parque Científico of Murcia
Parallellized three types of loop with different flops number, 3f, 8f and 19f, and execute in 4 systems:

- Rosebud: 4 Itanium dual core. 8 cores. Polytechnic University of Valencia.
- Hipatia: 2 Xeon E5462 quad-core. 8 cores. Polytechnic University of Cartagena.
- Ben: HP Integrity Superdome. 128 cores. Supercomputing Center of the Fundación Parque Científico of Murcia
- Arabi: 2 Xeon L5450 quad-core. 8 cores. Supercomputing Center of the Fundación Parque Científico of Murcia
Parallelization experiment

Parallellizated three types of loop with different flops number, 3f, 8f and 19f, and execute in 4 systems:

- **Rosebud**: 4 itanium dual core. 8 cores. Polytechnic University of Valencia.
- **Hipatia**: 2 Xeon E5462 quad-core. 8 cores. Polytechnic University of Cartagena.
- **Ben**: HP Integrity Superdome. 128 cores. Supercomputing Center of the Fundación Parque Científico of Murcia.
- **Arabi**: 2 Xeon L5450 quad-core. 8 cores. Supercomputing Center of the Fundación Parque Científico of Murcia.
Optimum core number for different systems and size problem

The diagram shows the optimum core number for different systems and size problem. The x-axis represents the number of nodes in the X axis of a square grid, while the y-axis represents the number of cores. Different systems such as Rosebud, Hipatia, Arabí, and Ben are plotted on the graph.
call omp_set_num_threads(14)
c$omp parallel
c$omp& private (i,j)
c$omp do

do i=2,nc
do j=1,nr
  if (npix(j,i).eq.1) then
    ud2(j,i) = dheddyvu(j,i)*((vd2atc(j,i)/h2atc(j,i)
1 -vd2atc(j,i-1)/h2atc(j,i-1))/gx2u(j,i)
2 +sphcur(j)*ud2(j,i)/h2atu(j,i))
  endif
end do
end do
c$omp end do
Carrying out various test with different size problem, we will be able to determine the optimum core number in each of the test carried out. This data will represent a cloud of point. Making a regression we can get the empirical equation.

\[ \text{Core} = \alpha + \beta X + \gamma Y \]
Carrying out various test with different size problem, we will be able to determine the optimum core number in each of the test carried out. This data will represent a cloud of point. Making a regression we can get the empirical equation.

\[ \text{Core} = \alpha + \beta X + \gamma Y \]
Carrying out various test with different size problem, we will be able to determine the optimum core number in each of the test carried out. This data will represent a cloud of point. Making a regression we can get the empirical equation.

\[ \text{Core} = \alpha + \beta X + \gamma Y \]
Auto-optimization strategies

**Test during installation**

During installation some experiments for representative problem sizes can be carried out to determine the number of cores on each loop. The optimum number of cores will be determined at running time for each loop and from the problem size (experimented at installation time) closest to that of the problem being solved.

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Optimum Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>1</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>400</td>
<td>2</td>
</tr>
<tr>
<td>600</td>
<td>3</td>
</tr>
<tr>
<td>800</td>
<td>5</td>
</tr>
<tr>
<td>1000</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Optimum Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>8</td>
</tr>
<tr>
<td>2500</td>
<td>8</td>
</tr>
<tr>
<td>3000</td>
<td>8</td>
</tr>
<tr>
<td>3500</td>
<td>8</td>
</tr>
<tr>
<td>4000</td>
<td>8</td>
</tr>
<tr>
<td>4500</td>
<td>8</td>
</tr>
</tbody>
</table>
Auto-optimization strategies

Adaptative code

Obtain the optimum number of cores for each loop at running time.

- In the first time step, get the time execution with 1 core (t1).
- In the second time step, get the time execution with 2 core (t2).
- If t2 > t1, the optimum core number = 1. Stop adaptation.
- If t2 < t1 continue with 3 core in the third time step, and compare if t3 > t2
Adaptative code

Obtain the optimum number of cores for each loop at running time.

- In the first time step, get the time execution with 1 core (t1).
- In the second time step, get the time execution with 2 core (t2).
- If t2 > t1, the optimum core number = 1. Stop adaptation.
- If t2 < t1 continue with 3 core in the third time step, and compare if t3 > t2
Adaptative code

Obtain the optimum number of cores for each loop at running time.

- In the first time step, get the time execution with 1 core ($t_1$).
- In the second time step, get the time execution with 2 cores ($t_2$).
  - If $t_2 > t_1$, the optimum core number = 1. Stop adaptation.
  - If $t_2 < t_1$ continue with 3 core in the third time step, and compare if $t_3 > t_2$
Adaptative code

Obtain the optimum number of cores for each loop at running time.

- In the first time step, get the time execution with 1 core ($t_1$).
- In the second time step, get the time execution with 2 core ($t_2$).
- If $t_2 > t_1$, the optimum core number $= 1$. Stop adaptation.
- If $t_2 < t_1$ continue with 3 core in the third time step, and compare if $t_3 > t_2$
Adaptative code

Obtain the optimum number of cores for each loop at running time.

- In the first time step, get the time execution with 1 core (t1).
- In the second time step, get the time execution with 2 core (t2).
- If $t_2 > t_1$, the optimum core number $= 1$. Stop adaptation.
- If $t_2 < t_1$ continue with 3 core in the third time step, and compare if $t_3 > t_2$
Mixed Strategies

Very important in system with a large number of cores. We know where to start to look for.

Empiric Time Model + Adaptative code
Values lower than 1 in the Y axis, indicate execution times better than those using the complete system.
Conclusions

- OpenMP can be easily used for development and auto-optimization of massively simulation codes for shared memory systems.
- The methodology has been analysed with COHERENS, but it can be used in other packages with a similar structure.
- The strategies explained is applicable to a wide range of systems.
- The use of all the core available don’t ensure us that we are getting the lower execution time.
- The tests carried out show the importance of having incorporated auto-optimization strategies for our parallel code which assure us that our computational time is near the optimum of our system.
- The next work to do is to integrate the methodology in COHERENS and analyse its application to other simulation packages.
Conclusions

- OpenMP can be easily used for development and auto-optimization of massively simulation codes for shared memory systems.
- The methodology has been analysed with COHERENS, but it can be used in other packages with a similar structure.
- The strategies explained is applicable to a wide range of systems.
- The use of all the core available don’t ensure us that we are getting the lower execution time.
- The tests carried out show the importance of having incorporated auto-optimization strategies for our parallel code which assure us that our computational time is near the optimum of our system.
- The next work to do is to integrate the methodology in COHERENS and analyse its application to other simulation packages.
Conclusions

- OpenMP can be easily used for development and auto-optimization of massively simulation codes for shared memory systems.
- The methodology has been analysed with COHERENS, but it can be used in other packages with a similar structure.
- The strategies explained is applicable to a wide range of systems.
  - The use of all the core avaible don’t ensure us that we are getting the lower execution time.
  - The tests carried out show the importance of having incorporated auto-optimization strategies for our parallel code which assure us that our computational time is near the optimum of our system.
  - The next work to do is to integrate the methodology in COHERENS and analyse its application to other simulation packages.
OpenMP can be easily used for development and auto-optimization of massively simulation codes for shared memory systems.

The methodology has been analysed with COHERENS, but it can be used in other packages with a similar structure.

The strategies explained is applicable to a wide range of systems.

The use of all the core available don’t ensure us that we are getting the lower execution time.

The tests carried out show the importance of having incorporated auto-optimization strategies for our parallel code which assure us that our computational time is near the optimum of our system.

The next work to do is to integrate the methodology in COHERENS and analyse its application to other simulation packages.
Conclusions

- OpenMP can be easily used for development and auto-optimization of massively simulation codes for shared memory systems.
- The methodology has been analysed with COHERENS, but it can be used in other packages with a similar structure.
- The strategies explained is applicable to a wide range of systems.
- The use of all the core available don’t ensure us that we are getting the lower execution time.
- The tests carried out show the importance of having incorporated auto-optimization strategies for our parallel code which assure us that our computational time is near the optimum of our system.
- The next work to do is to integrate the methodology in COHERENS and analyse its application to other simulation packages.
Conclusions

- OpenMP can be easily used for development and auto-optimization of massively simulation codes for shared memory systems.
- The methodology has been analysed with COHERENS, but it can be used in other packages with a similar structure.
- The strategies explained is applicable to a wide range of systems.
- The use of all the core available don’t ensure us that we are getting the lower execution time.
- The tests carried out show the importance of having incorporated auto-optimization strategies for our parallel code which assure us that our computational time is near the optimum of our system.
- The next work to do is to integrate the methodology in COHERENS and analyse its application to other simulation packages.
Thank’s a lot

Presentation done with: \LaTeX