Comparison of PGAS Languages on a Linked Cell Algorithm

M. Bauer\(^1\), C. Kuschel\(^1\), K. Sembritzki\(^2\), D. Ritter\(^1\)
April 11, 2013

\(^1\)Chair for System Simulation
\(^2\)Chair for Pattern Recognition
Motivation

- partitioned global address space (PGAS) languages promise to simplify development for parallel architectures
- goal: test PGAS languages by implementing a linked cell algorithm
- ease of use and performance evaluated
- compare PGAS implementations to a MPI parallel C version
Table of Contents

Linked Cell Algorithm

Implementation using PGAS Languages
   CAF
   UPC

Results

Conclusion and Outlook
Table of Contents

Linked Cell Algorithm

Implementation using PGAS Languages
  CAF
  UPC

Results

Conclusion and Outlook
Linked Cell Algorithm

- simulates interaction between particles (atoms/molecules)
- Assumption: only short range interactions, interaction for distances greater than $r_{cut}$ can be neglected
- choose cells with edge length $\geq r_{cut}$
- force calculation only between particles in current and neighboring cells
Lennard Jones Potential

Approximation of interaction potential between neutral atoms/molecules

\[
\left| \frac{F(r)}{r} \right| = \frac{24 \varepsilon}{r^2} \cdot \left( \frac{\sigma}{r} \right)^6 \cdot \left( 1 - 2 \cdot \left( \frac{\sigma}{r} \right)^6 \right)
\]
Basic Algorithm

Particle Structure:

```c
struct Particle {
    unsigned short type_id;
    double force [3];
    double position [3];
    double velocity [3];
};
```

Basic steps of Linked Cell algorithm:

1. **Compute Forces** for all particles due to particles in local and neighboring cells. *(local operation)*

2. **Position Update**: Time integration step that moves particles and updates velocity. Since particles can move across boundaries, this is a *non-local* operation.
Testcase: Raleigh-Taylor instability

- two particle types with different masses
- initially the heavier (yellow) particles are placed on top of the lighter (blue) particles
- scenario with 754,992 particles, into 138,240 cells ⇒ approx. 5.5 particles per cell
Table of Contents

Linked Cell Algorithm

Implementation using PGAS Languages
   CAF
   UPC

Results

Conclusion and Outlook
Partitioned global address space

Message Passing Model

PGAS Model

Thread → Thread → Thread

Memory → Memory → Memory

Thread → Thread → Thread

Memory
Co-Array Fortran (CAF)

program Hello_World

    integer :: i  ! Local variable
    character(len=20) :: name[*]  ! scalar coarray, one "name" for each image.

    ! Interact with the user on Image 1; execution for all others pass by.
    if ( this_image() == 1 ) then
      write( *, '(a)', advance='no' ) 'Enter your name: '
      read( *, '(a)' ) name
    end if

    ! Distribute information to other images
    do i = 2, num_images()
      name[i] = name
    end do
  end if

  sync all  ! Barrier to make sure the data have arrived.

  ! I/O from all images, executing in any order, but each record written is intact.
  write( *, '(3a,i0)' ) 'Hello ', trim( name ), ' from image ', this_image()

end program Hello_world
CAF Memory Layout

- cells store only indices into a global particle array
- global particle array stores position, velocity and force for local particles and position for neighbor particles
Communication

Processor $P$ has to receive:

- the position of all particles in those cells neighboring $P$’s domain
- position, velocity and net force of each particle that enters $P$’s domain.

```fortran
! sync with all neighbors
sync images(neighbor_indices)
do  i=1,26
   ! "push" communication with
   ! neighbor in direction i
   buffer[i] = sendbuffer(i)
end do
sync images(neighbor_indices)
```
Cascade High Productivity Language (Chapel)

- developed by Cray Inc.
- automatic parallelization done by compiler
- intuitive, easy to learn language
- problem with runtime library on test machine
- ⇒ no fair comparison possible
Unified Parallel C (UPC)

```c
#include <upc.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
    int i;
    for (i = 0; i < THREADS; ++i)
    {
        upc_barrier;
        if (i == MYTHREAD)
            printf("Hello world from thread: %d\n", MYTHREAD);
    }
    return 0;
}
```
UPC Data Distribution

shared int a[9];

```
<table>
<thead>
<tr>
<th>a[0]</th>
<th>a[1]</th>
<th>a[2]</th>
</tr>
</thead>
</table>
```

shared [3] int b[9];

```
|-------|-------|-------|
```
First variant: Shared global cell array

- goal: parallelizing without much programming effort
- 1D domain decomposition
- global shared domain array
- locking mechanism to serialize write access
First variant: Shared global cell array

- required very little programming effort (20-30 lines of code)
- suboptimal performance
- another UPC implementation with focus on performance was written

Optimization hint from UPC Manual:

"Use block copy instead of copying elements one by one with a loop"
Optimized UPC Implementation

- ghost layer based communication
- one message per timestep
- message passing: upc_memcpy
Table of Contents

Linked Cell Algorithm

Implementation using PGAS Languages
   CAF
   UPC

Results

Conclusion and Outlook
Overview: Programming Method and Effort

MPI in C (Effort: high)
- packed messages
- one message per timestep and neighbor

CAF (Effort: medium)
- global particle array
- packed messages

UPC “shared domain” (Effort: low)
- shared keyword to distribute domain
- communication by directly writing to neighbor cells (no packing)
- locked write access

UPC “messages” (Effort: high)
- comparable to MPI implementations
- upc_memcpy used instead of MPI_send/MPI_recv
Test Machine: SuperMUC Fat Node

- Testsystem: “Fat Node” of SuperMUC
- Westmere-EX Intel Xeon E7-4870 10C based node
- 4 Sockets a 10 cores ⇒ 40 cores
- one socket makes up a NUMA domain
- 256 GB of main memory
Test Machine: SuperMUC Fat Node

**UPC compilation:**
- Berkeley UPC Compiler (2.14.0)
- compiles first to C then uses GCC for final compilation
- to be comparable the MPI C version was also compiled with GCC instead of Intel
- “smp” network type

**CAF compilation:**
- built-in coarray support in Intel Fortran Compiler (12.1.6)
- communication type set to “-coarray=shared” for shared memory parallelization
Performance Results

- UPC: Message passing 2-3 times faster than “simple” UPC version
- UPC message passing has still an overhead compared to pure MPI
- CAF slower because of higher communication times
Computation- and Communication time

![Graph showing computation and communication times for different numbers of threads using MPI, UPC Message Passing, and CAF.](image)

- **Computation Time per Timestep [s]**
  - MPI
  - UPC Message Passing
  - CAF

- **Communication Time per Timestep [s]**
  - MPI
  - UPC Message Passing
  - CAF

April 11, 2013 | PGAS Languages Comparison on Linked Cell Algorithm
Table of Contents

Linked Cell Algorithm

Implementation using PGAS Languages
   CAF
   UPC

Results

Conclusion and Outlook
Conclusion and Outlook

Conclusion:
- PGAS languages allow for parallel programs to be written faster
- performance penalty in the range of factor 2-10 on test architecture
- before using PGAS language, check support on target architecture

Outlook:
- investigate PGAS languages using multiple nodes (distributed memory)
Thank you for your attention!

Questions?