

Is programming OpenMP really as easy as everyone says?

Kevin Roy

8th November 2002
UKHEC seminar
University of Edinburgh and
University of Manchester



THE UNIVERSITY
of MANCHESTER

Introduction

- See a few examples of codes I have worked with in OpenMP.
- See how I have gone about parallelising codes.
- See how not to go about parallelising codes 😊
- See real, yet simple, examples.
- Why some codes perform well with OpenMP.

Introduction

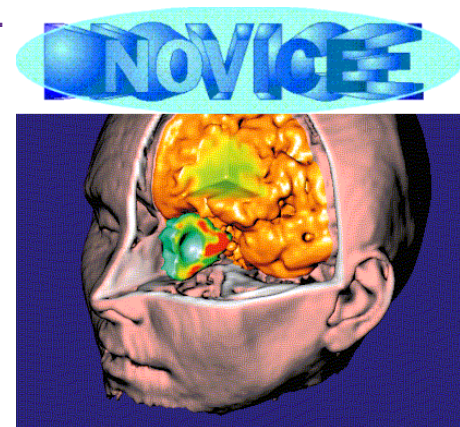
- Architecture used.
- NOVICE code
- SAMD code
- Matmul code
- Summary

Architecture Used

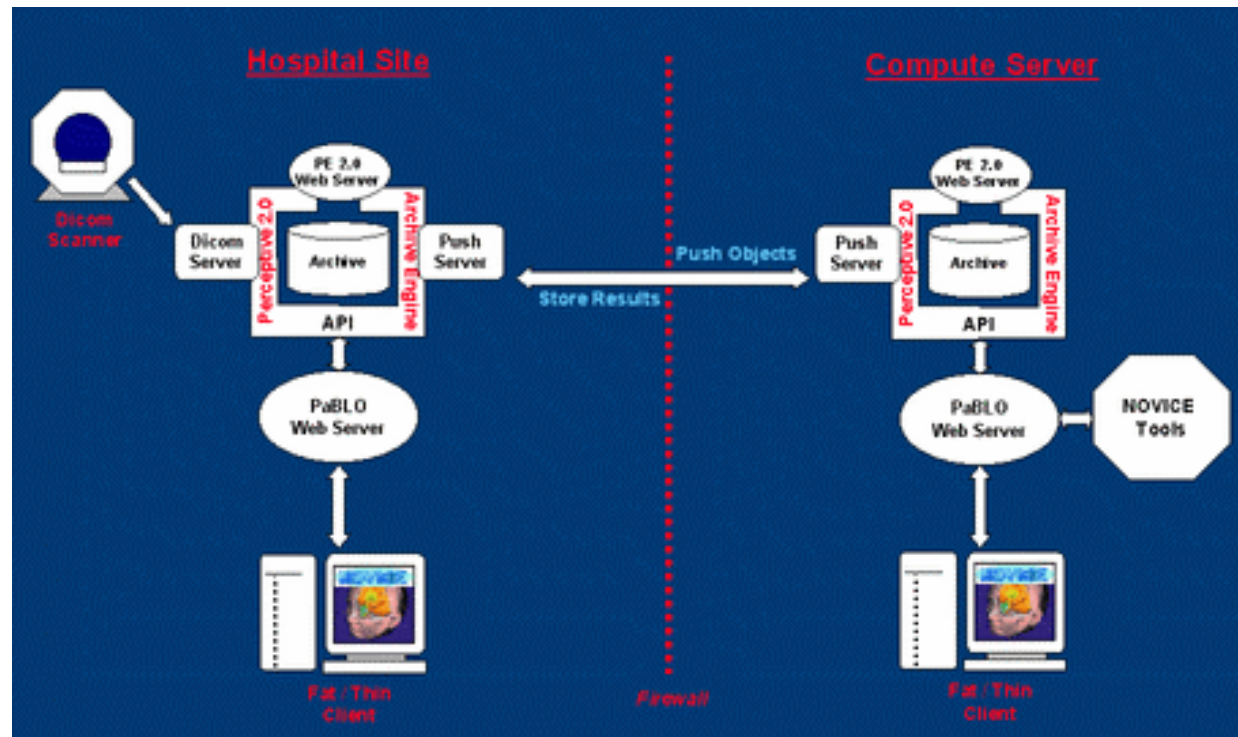
- Origin series.
- They are shared memory machines.
- At an architecture level they are distributed shared memory.
- In other words a cluster of small shared memory nodes.
- Important points in order to understand performance of programs:
 - Each processor has its own cache.
 - 2 or 4 processors share the memory.
 - Access to memory off node suffers a performance hit dependent on how far away it is.

NOVICE project

- NOVICE was a 3 year project funded under ESPRIT by the European Union
- The NOVICE project is developing a range of extensible Web-based visualization tools for medical applications that will work within a high-performance computing environment (HPCN).
- Data is scanned in at the hospital site which could be visualised or sent to the compute site for further analysis.
- This code was a wavelet transform in order to aid compression of the datasets.



NOVICE project



NOVICE Algorithm

- Uses unsophisticated wavelet transform (Haar)
- Takes sequence of data and performs operations on pairs.
- For sequence:
 - 1 4 -7 6
 - 2.5 -3 0.5 -13
 - 1.5 -3 2 -13
- To apply this in a multi-dimensional space, the algorithm is first applied in one direction along the whole plane then the other.

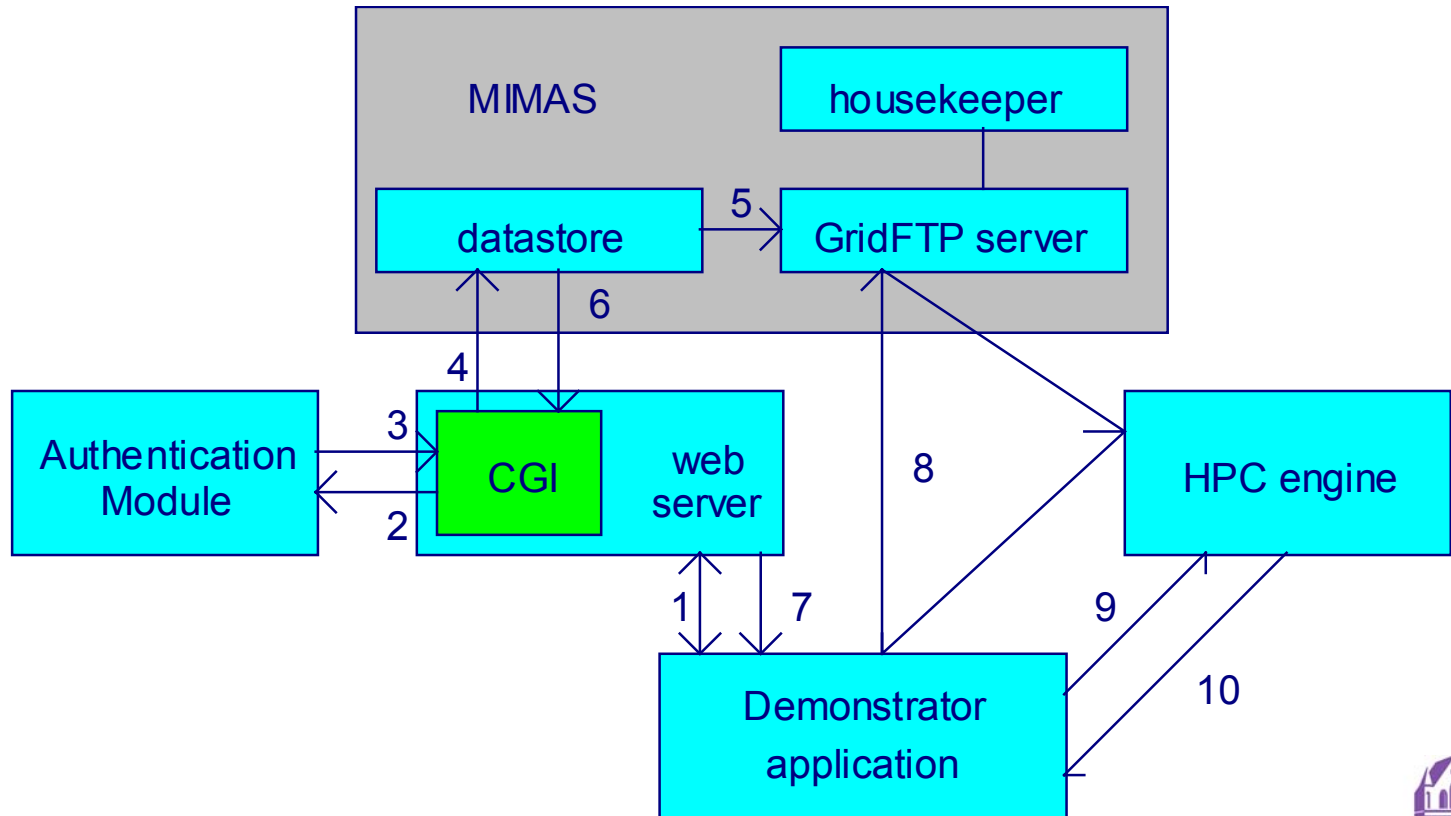
NOVICE - Algorithm

```
for (dc=0; dc<nc; dc++)
{  lower = upper;
  upper = i_hist[dc+1];
  for (level=0;level<logarithm(upper-lower);level++)
  {  offset = ipow(2,level);
    /* perform the required number of steps                               */
    for (i=lower; i<upper;i+=2*offset)
    {  for(j=jlmlower;j<jlimupper;j++)
      {  for(k=0;k<dms[2];k++)
        {  tmp = in[i][j][k] - in[i+offset][j][k];
          in[i][j][k]=(int) kfloor_div2(in[i][j][k]+
                                         in[offset+i][j][k]);
          in[offset+i][j][k]=tmp;
        }
      }
    }
  }
}
```

- The outer loop does not have enough iterations.
 - The purpose of this loop is to split the work up into powers of two blocks.
- The second loop introduces a dependency and there is also insufficient iterations (in general).
- Third loop was avoided because it looked too complicated,
 - Irregular accesses to data.
 - Also $1/2^n$ the amount of iterations dependent on level value.
 - In most cases not enough iterations.
- Had to choose fourth loop,
 - Has enough iterations.
 - Each iteration is completely independent.
 - In this case cache accesses are almost guaranteed to be independent.

- Part of an DTI ESRC demonstrator project of grid technology.
- Treats remote datasets of MIMAS web server as a grid resource.
- Treats HPC machine as a grid resource.
- Runs parallel application on HPC machine – data comes from MIMAS using gridftp and this is an automated process.
- HPC application started as a socio-economics Gauss program,
 - Needed to be converted to Fortran
 - Needed parallelising to show opportunities available.
 - Chose OpenMP as it was a small code and not much time could be put into parallelisation.

SAMD - topology



SAMD - application

The screenshot displays the SAMD application window with the following sections:

- Certification:** Includes buttons for 'grid-proxy-init' and 'grid-proxy-destroy'.
- Data Analysis:** Contains a 'Host Search' section with a 'Submit Job' button, an 'MDS Server / Port' field (ginfo.grid-support.ac.uk, 2135), 'OS Type' and 'Jobmanager Type' dropdowns, 'Number of Processors' (16), 'Other Criteria' field, and a 'Select Host' list containing 'ginfo.grid-support.ac.uk fork (default)'. Below this is a section for manual host entry with the text 'Or enter host & jobmanager manually' and the field 'fermat.cfs.ac.uk'.
- Data Acquisition:** Features a search interface with 'ONS' and 'CasWeb' tabs. The search term is 'gross domestic product'. Search parameters include 'Search Type' (Keyword), 'Periodicity' (Any), and 'Seasonality' (Any). It also has 'Start Year' and 'End Year' fields, and an 'Output format' dropdown set to 'Space separated'.
- Search Results:** A list of results including '053, IHYP, AU, Gross Do...', '188, IHYQ, QA, Gross Do...', '188, IHYR, QA, Gross Do...', '047, RVFD, AU, Statistical', '188, RVFD, QU, Statistica...', '054, YBEU, AU, Gross Do...', '188, YBEU, QA, Gross Do...', '054, YBEZ, AU, Gross do...', '188, YBEZ, QA, Gross do...', '054, YBGB, AU, Gross Do...', '188, YBGB, QA, Gross Do...', '054, YBHA, AU, Gross Do...', '188, YBHA, QA, Gross Do...', and '054, YBHH, AU, Gross Do...'. The entry '188, YBHH, QA, Gross Do...' is selected.
- Saved Results:** A list containing '188, YBHH, QA, Gross Dome' and '136, AMIJ, QU, Inter-bank :...'. The first entry is selected.
- Transferred Results:** A list containing '51445s_10078s_...'. The first entry is selected.
- Console Output:** Shows a log of search actions: 'Searching ONS databank...', 'Search complete', 'Searching ONS databank...', and 'Search complete'.

SAMD - OpenMP

- How easy was this?
 - It took about a week to do the OpenMP part.
 - This included me not understanding (or not reading) the OpenMP specification.
- Scalability
 - Very good scalability
 - Despite a very short run time.
- Timings
 - Gauss code - 40 minutes.
 - Fortran code – 5 minutes
 - 50 openMP threads – 10 seconds.
- However this was a very easy code to parallelise.

SAMD - considerations

- Outer most loop has insufficient iterations.
- One of the inner loops has 200 iterations (the best choice).
- Needs synchronisation
- Needs lots of local memory. Most of the memory required needs to be local.
- Code uses dynamical memory and Fortran 90 array syntax.
- Outside loops are significant enough to not want to re-setup threads.
- Personal target time of 10 seconds.
- Maintain style and structure of original gauss program as much as possible
- Time to achieve the scalability.

SAMD – The code

```
DO iind=1,2
  Calculate vectors y and z which come from input data set
  Put portions of y and z into ylag.
  DO mod=1,1
    Future proofing do loop
    Create final solution space
    DO d=1, dmax (15)
      DO gi=1, mg (200)
        DO zix=1,mc
          Lots of numeric calculations involving ylag but
            does not alter the input dataset.
          Inserts its part of answer into solution - scheduling
            very important here.
        END DO
      END DO
      Sort final solution space.
      Print part of solution
    END DO
  END DO
```

SAMD – The code

```
DO iind=1,2
    Calculate vectors y and z which come from input data set
    Put portions of y and z into ylag.

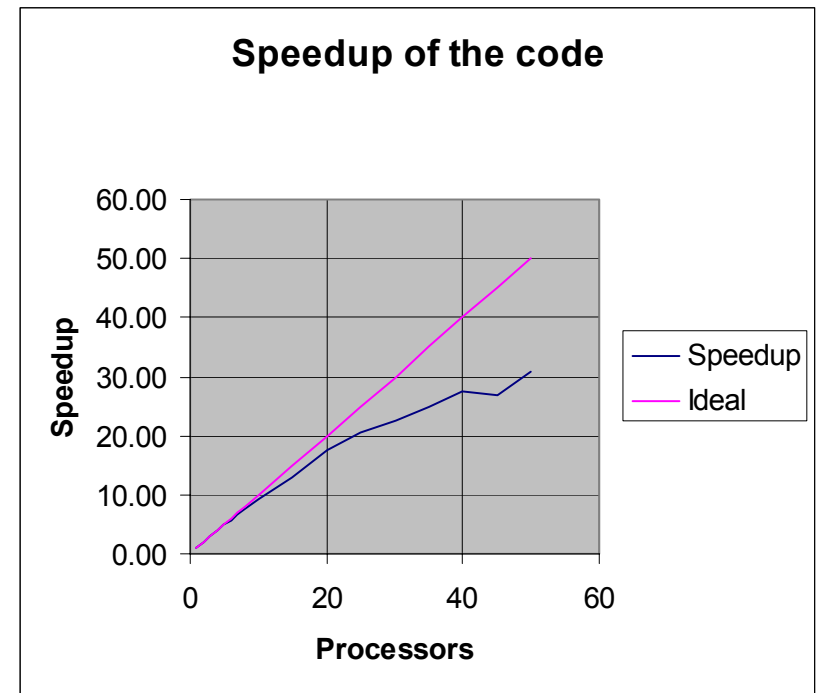
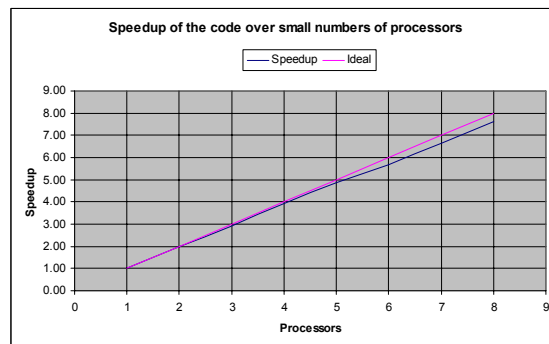
    DO d=1, dmax (15)
!$OMP DO (ylag, y and z are only shared variables)
        DO gi=1, mg (200)
            DO zix=1,mc (40)
                Lots of numeric calculations involving ylag but
                does not alter the input dataset.
                Inserts its part of answer into solution - scheduling
                very important here.
            END DO
        END DO
!$OMP END DO
        Sort final solution space.
        Print part of solution
    END DO
END DO
```

SAMD – The code

```
DO iind=1,2
  Calculate vectors y and z which come from input data set
  Put portions of y and z into ylag.
  !$OMP PARALLEL(ylag, y and z are only shared variables)
    DO d=1, dmax (15)
  !$OMP DO
    DO gi=1, mg (200)
      Dynamic memory allocations!!!!
      DO zix=1,mc (40)
        Lots of numeric calculations involving ylag but
        does not alter the input dataset.
        Inserts its part of answer into solution - scheduling
        very important here.
      END DO
    END DO
  !$OMP END DO
  !$OMP MASTER
    Sort final solution space.
    Print part of solution
  !$OMP END MASTER
  END DO
  !$OMP END PARALLEL
END DO
```

SAMD - performance

- Up to 20 processors the code is very scalable.
- On 45 threads some threads have 4 iterations others have 5 iterations.
- There are a few functions that are not implemented in OpenMP.
- At 50 threads the code only takes 9.5 seconds.



MATMUL

- Matrix multiplication is an important and computationally expensive (and interesting) task.
- Basic matmul operation no cache blocking, at the first stage.
- Started as a performance test for OpenMP, MPI and mixed mode programming model.
- MPI version uses cannon's algorithm.
- Learnt lots of lessons here.

MATMUL

- Very easy to insert OpenMP directives
- Surely choose outer loop and split iterations amongst threads?
- Provided we have no false sharing of A we should get very good results. Read accesses are not too important.
- But how well does the code scale
- Which is the best loop ordering using OpenMP.
- And how well does it compare to the MPI implementation.

MATMUL – loop ordering

- It is well known that some loop ordering is better than others.
- This is also the case for OpenMP
 - It is not the same loop ordering, though.
 - The outer loop is the one that is split up amongst the threads.
 - Much harder for the compiler to do anything with this loop.
 - Effects and reasoning are still under investigation.
 - Early results indicate that best choice is dependent on array size.

MATMUL - results

- Results appear very encouraging.
- Scalability also appears very good
- All effects can be explained
- So beat that MPI!!!

pes	1000	2000	4000	8000
1	2.833	23.09	643.88	5983
2	1.417	11.625	324.94	3013.9
4	0.717	5.817	163.95	1493.9
5	0.59	4.651	130.49	1203.99
8	0.381	2.961	82.41	753.12
10	0.307	2.366	65.34	602.64
16	0.34	1.53	42.86	379.65

MATMUL - MPI

- Well it does in some cases.
- 4000 by 4000 element array only needs 4 processors to begin beating the OpenMP.
- On an 8000 by 8000 element array we need 16 processors to see this super linear effect.

Pes	1000	2000	4000	8000
1	2.87	23.27	780.03	5983
4	0.84	6.35	49.02	1584
16	0.258	1.736	12.76	97.98

MATMUL – mixed mode

- We have seen that OpenMP is much better on smaller matrix sizes.
- MPI is much better on larger matrix sizes.
 - The decomposition in to a number of smaller chunks allows us to have the cache effects of a much smaller system.
 - E.g., a 2x2 processor grid applied to 4000x4000 problem decomposes this into 2 lots of 2000x2000 problem which is already known.
- The smaller problem on each node is perhaps best tackled by OpenMP!
- This is indeed the case although I do not have the results ☹

Acknowledgements

- Neil Stringfellow
- James Perrin
- Stephen Pickles
- Etc etc

Manchester Computing

Supercomputing, Visualization & e-Science

SVE @ Manchester Computing

World Leading Supercomputing Service,
Support and Research

***Bringing Science and
Supercomputers Together***

www.man.ac.uk/sve
Kevin.Roy@man.ac.uk



THE UNIVERSITY
of MANCHESTER