Abstract

Vendors of high performance computers are converging towards an architecture based on tightly-coupled clusters of shared memory nodes. A new style of parallel programming is required to take full advantage of the available computing power, in order to achieve the best scalability.This new style uses a mixed model of thread-based parallelism and message passing. The former, which has very low overhead, is used within each shared memory node, while the latter is needed for inter-node communications. In this work, the mixed-model method was applied to add a new level of parallelism to the 3D gyrokinetic code developed at PPPL to study microturbulence in magnetized plasmas[1]. Various performance issues are discussed as well as scalability and implementation.

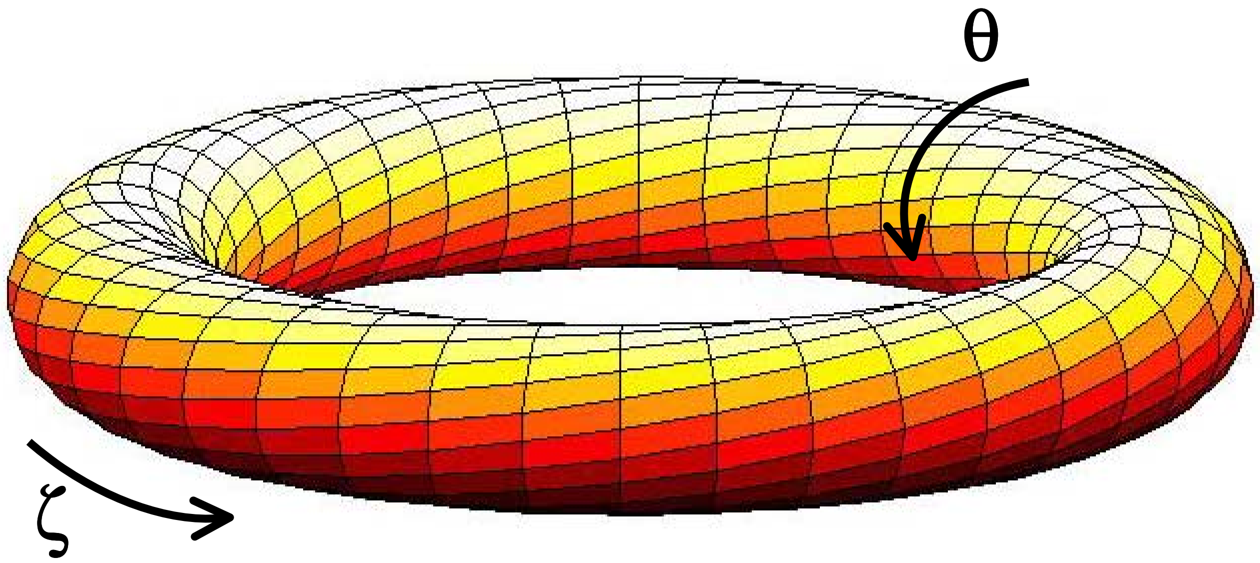
[1] Z. Lin, T.S. Hahm, W.W. Lee, W.M. Tang, and R..B. White, Science 281, 1835 (1998). This work is supported by DOE Contract No. DE-AC02-76CH03073 (PPPL), and in part by the Numerical Tokamak Turbulence Project.

Gyrokinetic Toroidal Code

- Description:
  - Particle-in-cell code (PIC)
  - Gyrokinetic simulation of microturbulence [*Lee, 1983*]
  - Fully self-consistent
  - Uses magnetic coordinates (ψ,θ,ζ) [*Boozer, 1981*]
  - Guiding center Hamiltonian [*White and Chance, 1984*]
  - Non-spectral Poisson solver [*Lin and Lee, 1995*]
  - Low numerical noise
  - Full torus (global) simulation

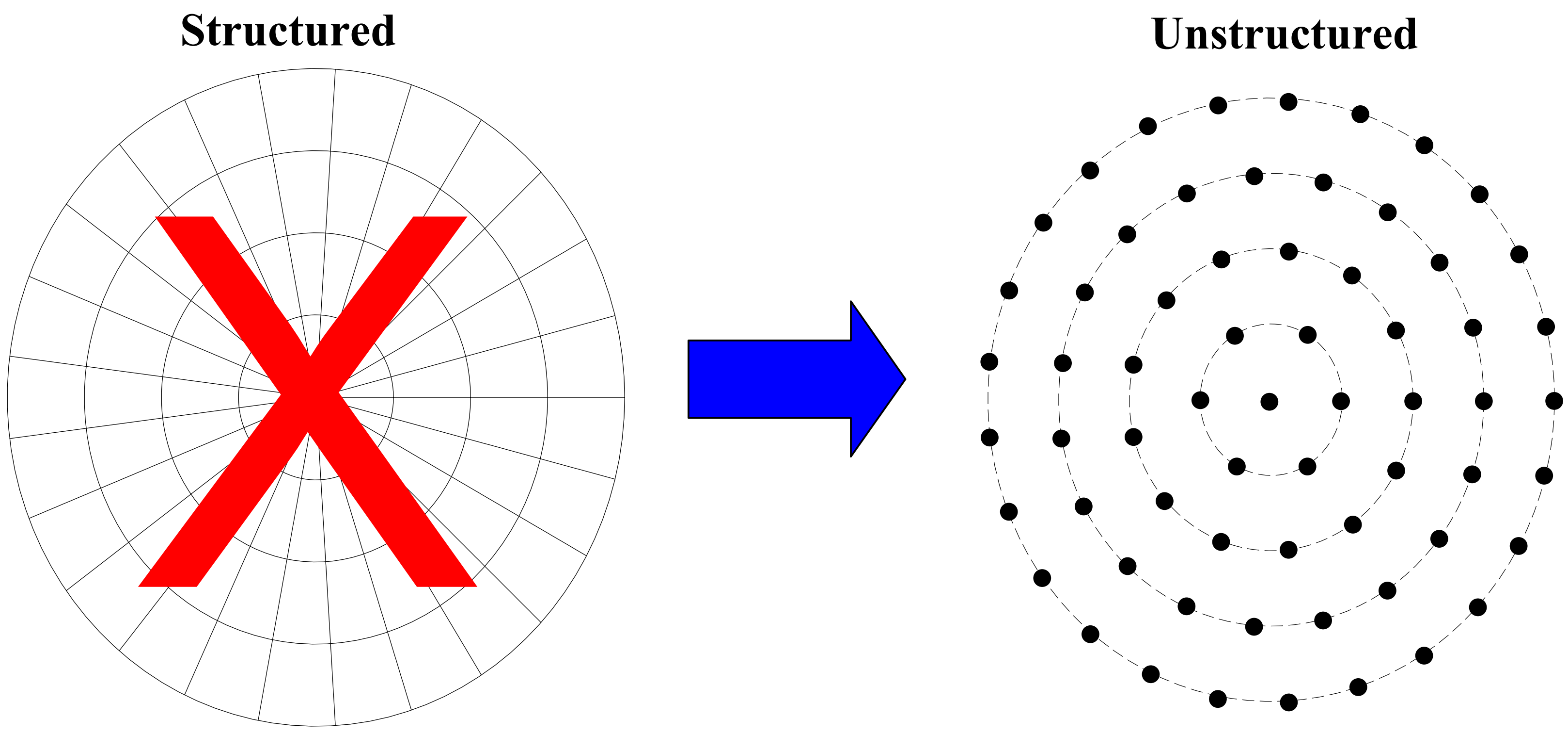
Geometry & Coordinates

- Field-line following coordinates
  - $(\psi,\alpha,\zeta) \quad \Rightarrow \quad \alpha = \theta - \zeta/q$
  - larger time step: no high order k|| modes
  - order of magnitude saving of computer time



Poloidal Grid: unstructured mesh

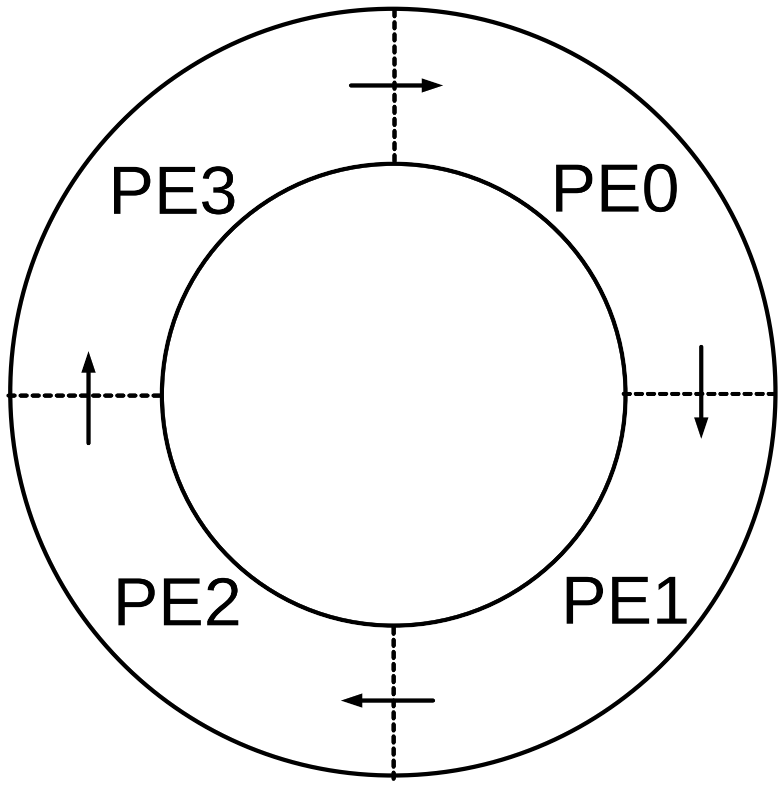
- Gives uniform resolution (constant volume grid cell)
- Smaller number of grid points to describe system



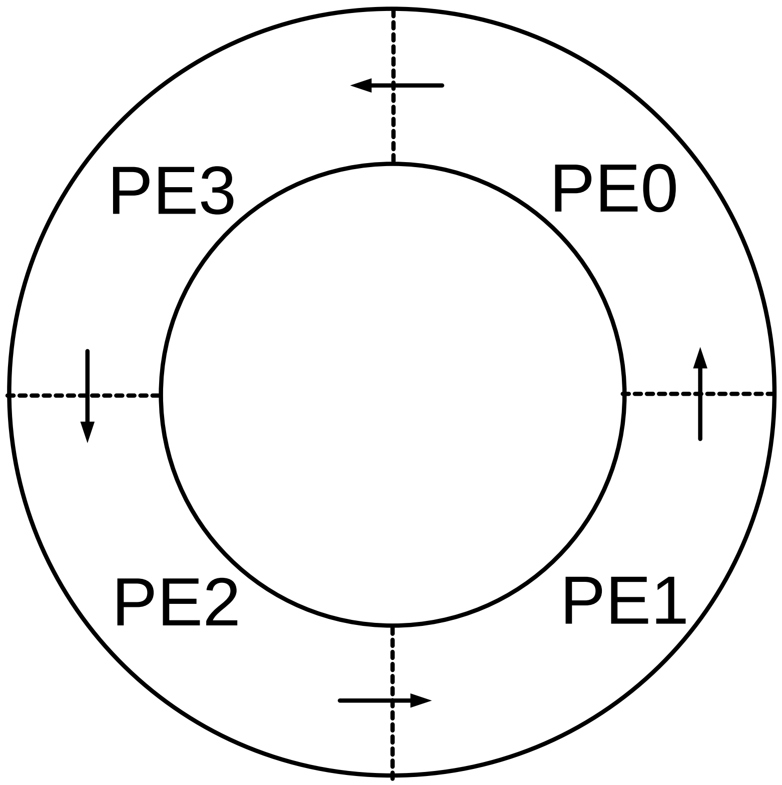
Original Code Parallelization

- Domain decomposition:
  - each processor holds a section of the toroidal geometry
  - each particle is assigned to a processor according to its position
- Communication between processors is done with Message Passing Interface (MPI)
- Initial memory allocation is done locally on each processor to maximize efficiency
- Uses standard MPI calls so the code runs on most parallel computers

Efficient Communication Scheme

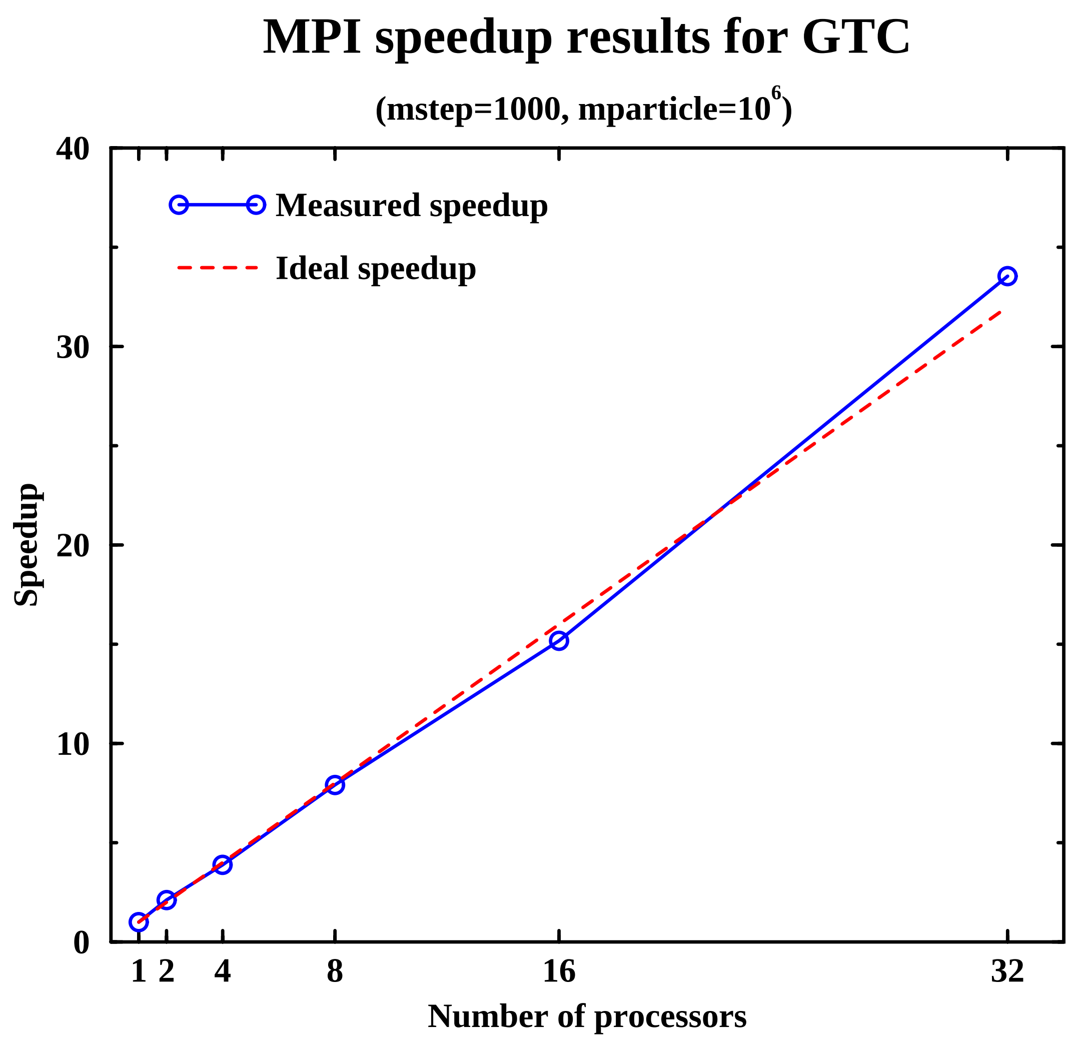


step 1



step 2

MPI Scaling for GTC



New Level of Parallelization?

- <u>MPI scaling is almost perfect so WHY do we need to add a new level of parallelization???</u>
  - Landau damping puts a limit on the number of modes in the direction parallel to the **B** field (toroidal direction):
    - ⇒ no high order k|| modes!!!
  - The useful resolution along the field lines is about 64 grid points: a larger number of grid points would not add any relevant physics
  - Domain decomposition in the toroidal direction is thus limited to around 64 domains (= 64 processors)
  - To use a greater number of processors efficiently, we need to add a level of parallelization...

PPPL
PRINCETON PLASMA PHYSICS LABORATORY

Why Mixed-Mode?

- VERY easy to implement at loop level
- Most high performance computers now have a shared memory architecture (SGI Origin 2000, IBM SP, CRAY J90,...)
- Low overhead thread-based parallelism
- Low latency/high bandwidth communications between threads via shared memory
- Can be used within each MPI process
- Save memory: no need for ghost cells...

Approach to loop-level parallelization

- STEPS:
  - Identify loops appropriate for parallelization
  - use analysis tools to determine the percentage of work done inside those loops (work which will be done in parallel)
    - ⇒ examples of such tools on the SGI Origin 2000 (IRIX) are SPEEDSHOP Pro and PERFEX
  - use Amdahl’s law to determine the maximum theoretical speedup to expect
  - Decide if it is worth the effort...

Timing of GTC's subroutines

- Timing of the most important subroutines by using SPEEDSHOP pro
- **<u>88%</u>** of the calculation time is spent inside charge, pusher, and poisson

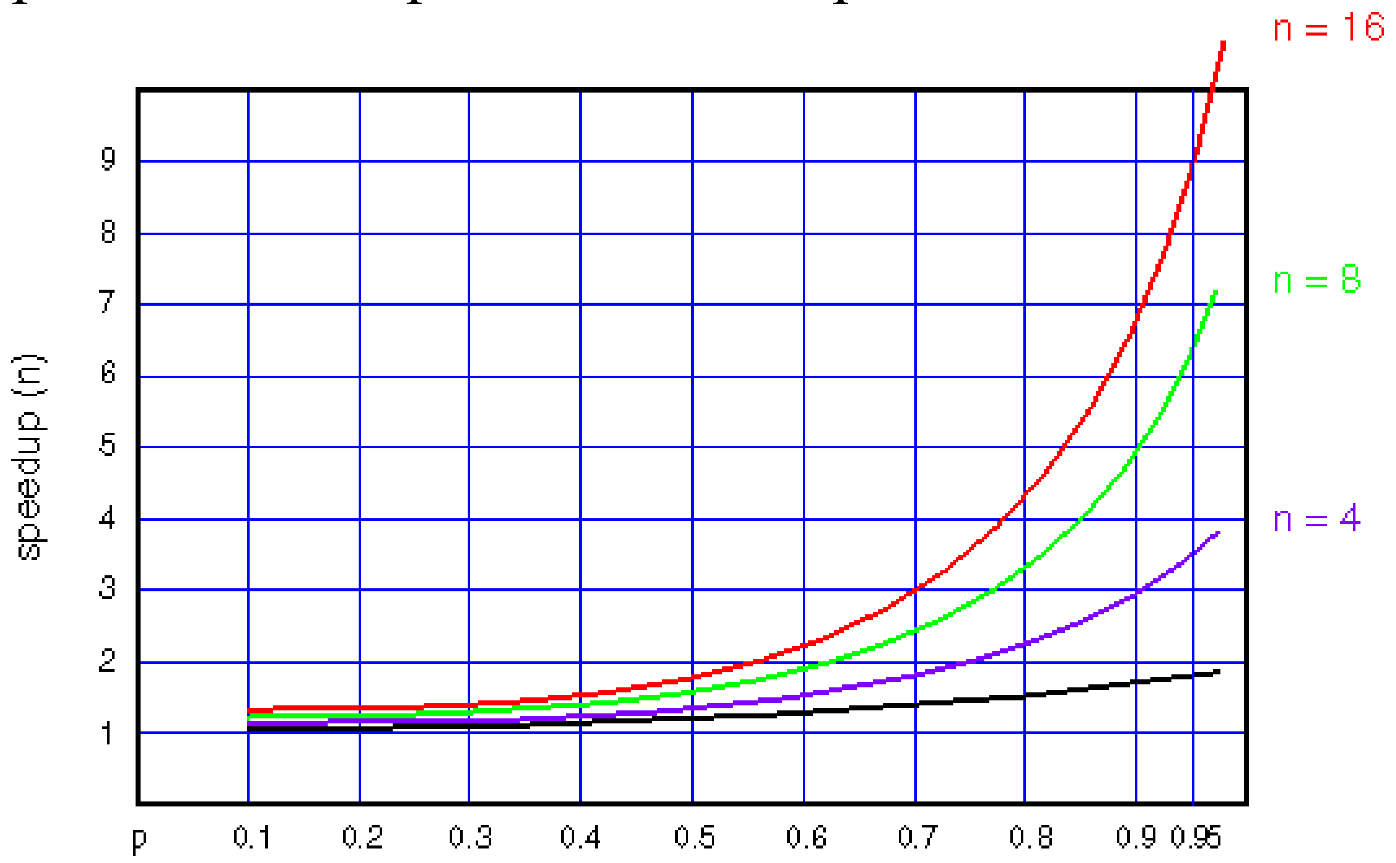
```
---------------------------------------------------------------
Function list, in descending order by time
---------------------------------------------------------------
 [index]      secs      %    cum.%    samples  function
     [1]  4938.945  40.9%    40.9%    4938945  charge
     [2]  4852.051  40.2%    81.1%    4852051  pusher
     [3]   828.398   6.9%    88.0%     828398  poisson
     [4]   404.091   3.3%    91.4%     404091  shift
     [5]   395.813   3.3%    94.6%     395813  __libm_rcis
     [6]   232.287   1.9%    96.6%     232287  __expf
     [7]   198.840   1.6%    98.2%     198840  smooth
     [8]   162.687   1.3%    99.6%     162687  field
     [9]    14.336   0.1%    99.7%      14336  load
    [10]    12.155   0.1%    99.8%      12155  memcpy
    [11]     3.497   0.0%    99.8%       3497  C06FAY
    [12]     2.602   0.0%    99.8%       2602  poisson_initial
```

Refine timings to loop-level

- Charge:
  - 1 loop over "mp" (number of particles in the domain) : 98.7% of the subroutine work
- Pusher:
  - 3 loops over "mp", 2 of them contribute at every time step: 81 - 95% of the work
- Poisson:
  - 1 loop over number of toroidal grid points in domain: 99.5% of work in subroutine
- TOTAL PARALLEL WORK IS ABOUT 84% ON AVERAGE BUT CAN GO UP TO 87%

Amdahl's Law

- Speedup(n) = 1/[(p/n) + (1-p)]   where n is the number of processors and p the fraction of parallel work



OpenMP parallel directives

- What is OpenMP?
  - "The OpenMP Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms. Jointly defined by a group of major computer hardware and software vendors, OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer" (from http://www.openmp.org)

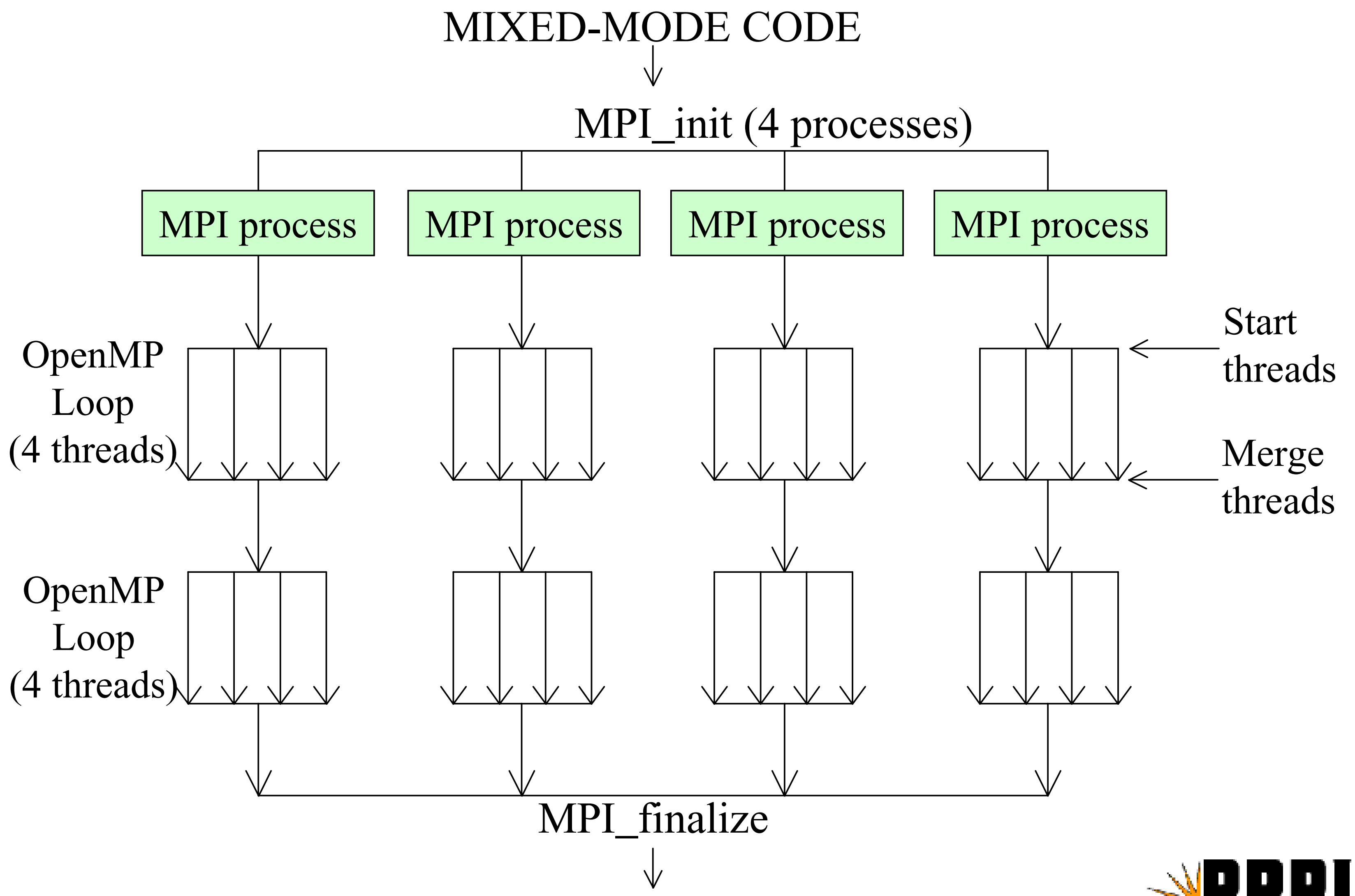
- We used the OpenMP directives to parallelize the biggest loops (4 of them) in GTC

OpenMP example of loop-level parallelization

- Subroutine "charge": loop over the particles inside an MPI domain

```
!$omp parallel do private(psitmp,thetatmp,zetatmp,weight,&
!$omp&rhoi,r,ip,jt, ipjt,wz1,kk,wz0,larmor,rdum,ii,wp1,wp0,&
!$omp& tflr,im,tdum,j00,wt10,wt00,j01,wt11,wt01,ij)
  do m=1,mp
     psitmp=phase(1,m)
     thetatmp=phase(2,m)
     zetatmp=phase(3,m)
     weight=phase(5,m)
     rhoi=phase(6,m)*g_inv
     ...
  enddo
```

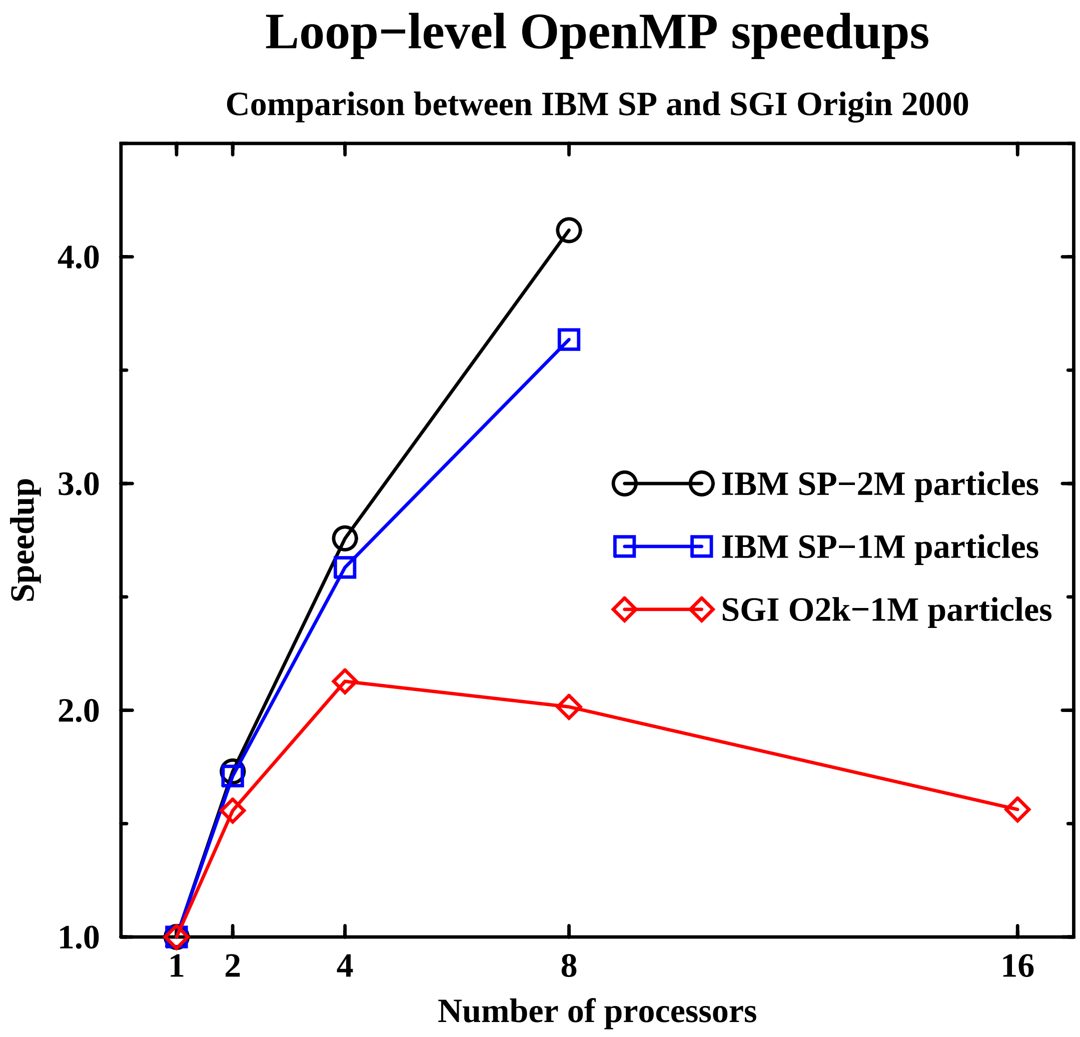
How does it work?



What we did…

- We parallelized the 4 loops with simple OpenMP directives
- Compiled and run on 2 different shared memory supercomputers:
  - 64-processor SGI Origin 2000 “Hecate” at Princeton
    - ⇒ 32 nodes with 2 processors/node but fully shared memory
  - 1,152-processor IBM SP “Blue Horizon” at the San Diego Supercomputing Center (SDSC)
    - ⇒ 144 nodes with 8 processors/node
- Ran a fixed-size problem with 1 and 2 million particles, 1 MPI process, and 1, 2, 4, 8 OpenMP threads (also 16 threads on the Origin 2000)

OpenMP Scaling for GTC



Comparison with Amdahl's law

- Our timing analysis told us that the maximum fraction of parallel work in the 4 main loops was 87%. How do the best scaling compare with Amdahl's law?

Number of processors	IBM SP with 2M particles	Amdahl's law with p=0.87
2	1.7	1.8
4	2.8	2.9
8	4.1	4.2

PPPL
PRINCETON PLASMA PHYSICS LABORATORY

Results of OpenMP scaling

- With 2 million particles, the IBM SP gives a scaling very close to the maximum theoretical scaling given by Amdahl’s law.
- With 1 million particles, the SP results are not quite as good but still excellent.
- The SGI Origin 2000 has very bad scaling when we use more than 2 processors…

WHY???

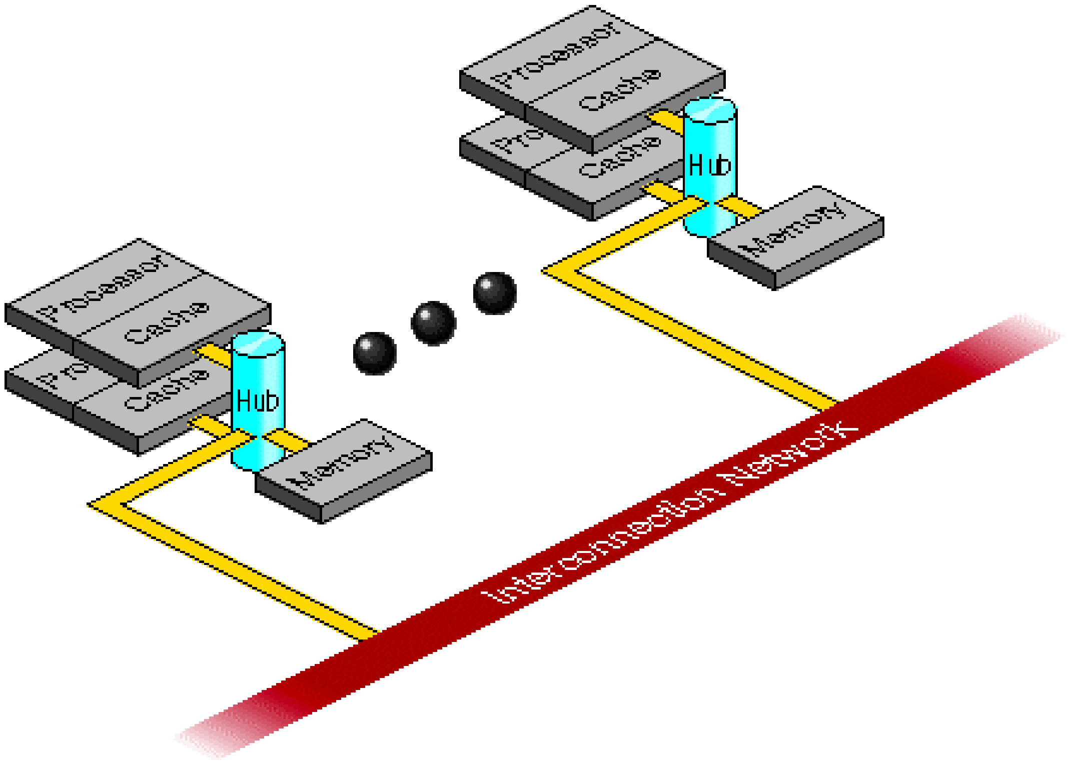
PPPL
PRINCETON PLASMA PHYSICS LABORATORY

Why is the IBM SP so much better?

- Each SP node has a true SMP "Symmetric Multiprocessing" architecture:
  - each processor on the node has the same access to the local memory (through 2 levels of cache)
  - all the communications between threads are done through local memory on the node
- The Origin 2000 has a NUMA "Non Uniform Memory Access" architecture
  - each processor can access (and address) the memory of ALL the nodes on the computer
  - The user has very little control over communications between threads (= non local)

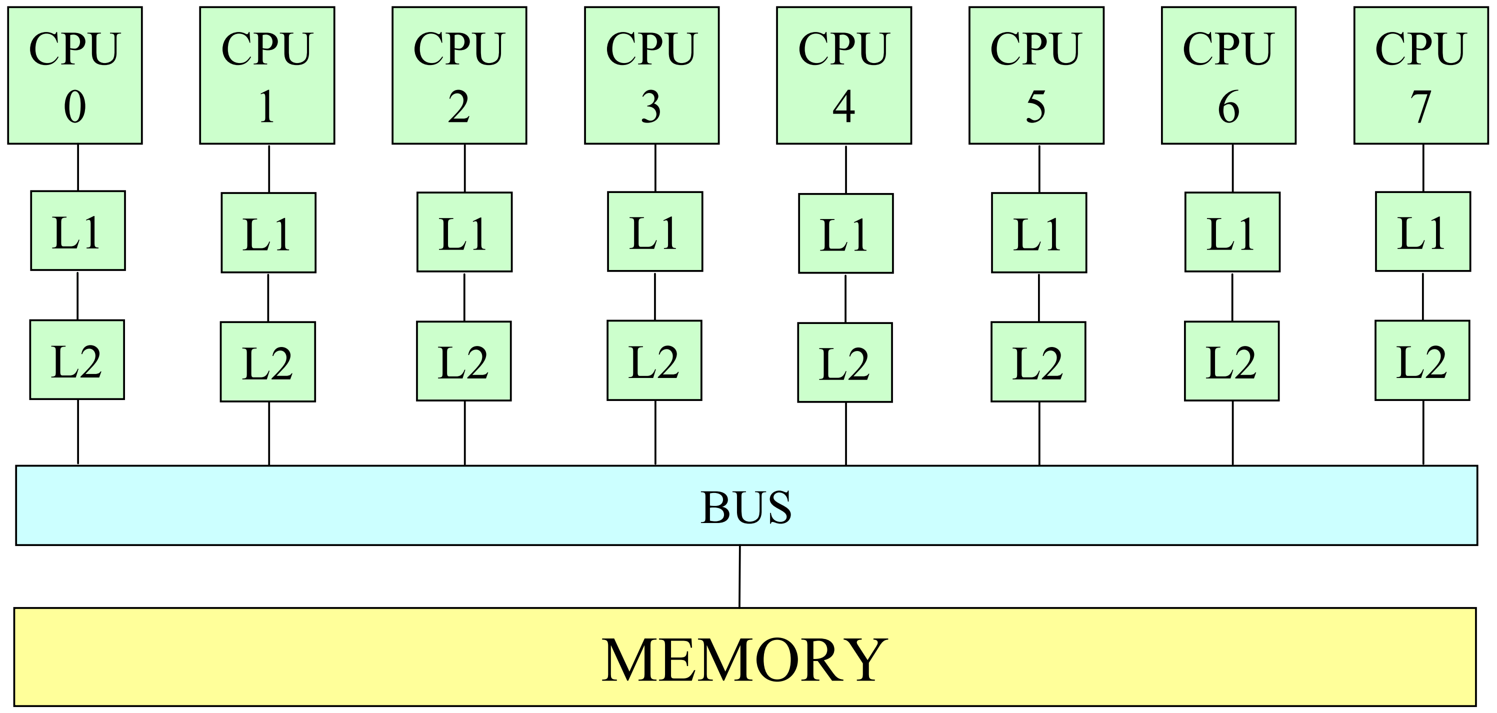
Origin 2000 NUMA architecture

- The operating system takes care of communications between nodes. As far as the user is concerned, the Origin 2000 is one big shared memory machine.



IBM SP true SMP nodes

- Each of the 8 processors on a SMP node of Blue Horizon has the same link to the local memory but cannot address the memory of other nodes



Conclusions

- Mixed-mode parallelization is a good way to take advantage of the shared memory nodes that are now used in most MPP computers.
- Loop-level parallelism is easy to implement with the OpenMP API but very “fine-grained”.
- Parallel speedup is limited by the amount of work contained in the loops.
- The speedup agrees with Amdahl’s law as long as the threads are accessing only their caches and the local memory.

PPPL PRINCETON PLASMA PHYSICS LABORATORY