Experiences benchmarking and optimizing GTC on High Performance Computers

Stéphane Ethier
Princeton Plasma Physics Laboratory

NERSC Users’ Group meeting
June 2006

Work Supported by DOE Contract No.DE-AC02-76CH03073 and by the DOE SciDAC Center for Gyrokinetic Particle Simulation of Turbulent Transport in Burning Plasmas.
The Gyrokinetic Toroidal Code

- 3D particle-in-cell code to study microturbulence in magnetically confined fusion plasmas.
- Solves the gyro-averaged Vlasov equation.
- Gyrokinetic Poisson equation solved in real space.
- Low noise $\delta f$ method.
- Global code (full torus as opposed to only a flux tube).
- Massively parallel: typical runs done on 1024 processors.
- Electrostatic approximation with adiabatic electrons.
- Nonlinear and fully self-consistent.
- Written in Fortran 90/95
- Well optimized for superscalar processors
Particle-in-cell (PIC) method

- Particles sample distribution function.
- The particles interact via a grid, on which the potential is calculated from deposited charges.

### The PIC Steps

- “SCATTER”, or deposit, charges on the grid (nearest neighbors)
- Solve Poisson equation
- “GATHER” forces on each particle from potential
- Move particles (PUSH)
- Repeat…
Charge Deposition for charged rings: 4-point average method

Point-charge particles replaced by charged rings due to gyro-averaging

Charge Deposition Step (SCATTER operation)

Classic PIC 4-Point Average GK (W.W. Lee)
Quasi-2D structure of potential

- Fast particle motion along the magnetic field lines leads to a quasi-2D structure in the electrostatic potential
- Poisson equation needs only to be solved on 2D poloidal plane
GTC mesh and geometry

\[(\Psi, \alpha, \zeta) \Rightarrow \alpha = \theta - \zeta/q\]

Saves a factor of about 100 in CPU time

Poloidal plane (cross-section)
unstructured mesh

Field-line following coordinates
Original parallel model in GTC:
1D toroidal domain decomposition

- Uses Message Passing Interface (MPI)
- Each MPI process holds a toroidal section
- Most of the communications due to particles moving in and out of the toroidal domains (10% of particles at each time step)
- Efficient “ring-type” communication when moving particles.
- Scales perfectly but limited to about 64 or 128 domains due to (Landau) damping of shorter wavelength modes.
Scaling of original version of GTC

Y-axis: the number of particles (in millions) which move 1 step in 1 second
Then came Seaborg…

- The arrival of the IBM SP Power 3 Seaborg at NERSC opened new possibilities for higher performance.
- First step: port GTC from the T3E to the SP and optimize single processor performance
  - Larger memory allowed us to reuse calculations done in the charge deposition subroutine
- The Symmetric Multi-Processing (SMP) nodes of the IBM SP gave an easy path to higher concurrency for GTC: Shared memory programming
- With 16 processors per node, Mixed-model MPI+OpenMP would allow GTC to run on 1,024 processors instead of only 64
New level of parallelism in GTC: Loop-level

MPI_init

MPI process

OpenMP Loop

Start threads

OpenMP Loop

Merge threads

MPI finalize
Why loop-level parallelism?

- **VERY EASY TO IMPLEMENT…**
- Although one has to watch out for potential conflicts between threads (processors) trying to write to the same memory location at the same time
  - Easily solved by using thread-private copies of conflicting arrays
- 85% of the work in GTC reside in 4 loops over the number of particles on each MPI process.
- Adding the other loops pushes the amount of computational work in parallel loops beyond 90%.
- The bigger the loops (problem size), the more efficient is the calculation (we saw 98% on large simulations).
OpenMP example of loop-level parallelism

- Simple but powerful OpenMP directives

```c
!$omp parallel do private(psitmp,thetatmp,zetatmp,weight,&
!$omp&rholo,r,ip,jt, ipjt,wzl,kk,wz0,larmor,rdum,ii,wp1,wp0,&
!$omp& tflr,im,tddum,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
```
Mixed-model MPI+OpenMP lead to first ITER-size simulations

• With mixed-model a single MPI process is assigned to each SMP node on Seaborg
  – Large amount of memory per MPI process (32 GB/proc!)
  – Had to wait for 64-bit MPI to access it though…

• Allowed size scaling study of turbulent transport in tokamaks, including ITER size:
  – 1 billion particles
  – 125 million grid points
  – 1,024 processors
  – largest GTC run at the time
Interesting benchmark of OpenMP on IBM SP and SGI Origin 2000

- SGI O2k has really only 2 processors that share local memory symmetrically.
- The NUMA architecture performs poorly unless processor placement is used.
- The symmetric memory access for the processors on the IBM SP node is ideally adapted to the mixed-model algorithm.

Loop-level OpenMP speedups
Comparison between IBM SP and SGI Origin 2000
Seaborg allows GTC to routinely run on 1000+ processors.
Then came the others…

- Newer, bigger, and faster computers continuously emerge.
- It prompted a renewed interest in vector processing.
- Cray introduced the X1 vector machine soon after.
- I was invited to participate in a study of modern vector architectures compared to current superscalar ones such as the IBM SP.
- The study was lead by Dr Leonid Oliker of the Future Technologies Group at LBL.
GTC vectorization work

• Started on the single-node NEC SX-6 at ARSC
• Porting GTC was very easy although the first tests on a single processor gave a very low performance
• Real work starts: profiling, vectorizing, optimizing, test, and... repeat several times
• Multi-processor optimization done on to the Earth Simulator and CRAY X1
The charge deposition step (scatter operation) writes to the charge accumulation array in a random fashion (particle positions are random), producing dependencies and memory conflicts whenever 2 or more particles have a common neighboring grid point → this prevents vectorization.

In 1D, the charge deposition step with linear interpolation looks like this:

```fortran
do i=1,nparticles
    x = particle_position(i)
    ix_grid = int(x)
    dx = x - real(ix_grid)
    charge(ix_grid) = charge(ix_grid)+q*(1-dx)  \ Indirect addressing!
    charge(ix_grid+1) = charge(ix_grid+1)+q*dx \ Potential Conflicts
end do
```
Avoiding memory dependencies: The work-vector method (Nishiguchi ‘85)

Example of loop with indirect addressing similar to charge deposition:

```fortran
DO i=1,np
    charge(ix(i)) = charge(ix(i)) + q(i)
END DO
```

Fully vectorizable loop using multiple copies (vector length of 256):

```fortran
ALLOCATE(charge_tmp(256,ngrid))
DO i=1,np,256
    DO j=1,min(256,np-i+1)
        charge_tmp(j,ix(i+j-1)) = charge_tmp(j,ix(i+j-1)) + q(i+j-1)
    END DO
END DO
DO i=1,256
    DO ig=1,ngrid
        charge(ig) = charge(ig) + charge_tmp(i,igrid)
    END DO
END DO
```

Uses 256*ngrid*sizeof(charge_tmp) of extra memory! (can be 1GB)
Loop-level multithreading competes directly with vectorization

- Each Earth Simulator node has 8 vector processors sharing 16 GBytes of memory, allowing us to use GTC’s mixed-model MPI+OpenMP.
- However, loop-level work splitting with OpenMP reduces the number of loop operations, which in turn degrades vector efficiency → Lower performance
- Charge deposition loop with OpenMP requires private copies of the grid array for each processor on the node.
- Combined with the 256 copies of the same grid array needed for vectorization, the loop-level OpenMP requires too much memory.
Cache-less memory access issues on the SX-6 and ES

- Better memory access is the secret to higher performance
- True for STORING to memory as well as FETCHING from it!

```fortran
  do m=1,mi
    psitmp=zion(1,m)
    thetatmp=zion(2,m)
    zetatmp=zion(3,m)
    rhoi=zion(6,m)*smu_inv
    r=sqrt(2.0*psitmp)
    ip=max(0,min(mpsi,int((r-a0)*delr+0.5)))
    jt=max(0,min(mtheta(ip),int(thetatmp*pi2_inv*delt(ip)+0.5)))
    ipjt=igrid(ip)+jt
    wz1=(zetatmp-zetamin)*delz
    ...
```

Repeatedly accessing the same memory bank before the bank busy time is over from the last access leads to poor memory performance!

Duplicate small arrays like “igrid” and “mtheta”: !$duplicate 37% improvement on chargei, but uses even more memory…
Vector performance of main routines on the Earth Simulator

**ORIGINAL CODE BEFORE MODIFICATIONS:**

<table>
<thead>
<tr>
<th>PROG.UNIT</th>
<th>EXCLUSIVE TIME[sec] ( % )</th>
<th>MFLOPS</th>
<th>V.OP RATIO</th>
<th>AVER. V.LEN</th>
<th>BANK CONF</th>
</tr>
</thead>
<tbody>
<tr>
<td>chargei</td>
<td>282.677 ( 54.4)</td>
<td>62.0</td>
<td>0.65</td>
<td>98.1</td>
<td>0.0000</td>
</tr>
<tr>
<td>pushi</td>
<td>125.211 ( 24.1)</td>
<td>320.1</td>
<td>67.51</td>
<td>196.8</td>
<td>4.3336</td>
</tr>
<tr>
<td>poisson</td>
<td>57.878 ( 11.1)</td>
<td>418.9</td>
<td>94.26</td>
<td>107.2</td>
<td>0.3158</td>
</tr>
</tbody>
</table>

Note: the 2 tests do not have the same number of time steps so the times are different

**CODE AFTER MODIFICATIONS TO CHARGEI, PUSHI, POISSON:**

<table>
<thead>
<tr>
<th>PROG.UNIT</th>
<th>EXCLUSIVE TIME[sec] ( % )</th>
<th>MFLOPS</th>
<th>V.OP RATIO</th>
<th>AVER. V.LEN</th>
<th>BANK CONF</th>
</tr>
</thead>
<tbody>
<tr>
<td>chargei</td>
<td>89.924 ( 33.3)</td>
<td>1314.3</td>
<td>99.65</td>
<td>248.1</td>
<td>6.5002</td>
</tr>
<tr>
<td>pushi</td>
<td>93.877 ( 34.7)</td>
<td>2426.6</td>
<td>99.38</td>
<td>255.9</td>
<td>8.8139</td>
</tr>
<tr>
<td>poisson</td>
<td>26.239 (  9.7)</td>
<td>918.1</td>
<td>99.71</td>
<td>252.7</td>
<td>3.2485</td>
</tr>
</tbody>
</table>

Total = 1.412 Gflops per proc
GTC on the CRAY X1/X1E

- Must deal with multi-streaming on top of vectorization
- Same vectorizations apply.
- Easier to prevent vectorization of small inner loops
- Also needs the work-vector method with the same dimensions of 256 in MSP mode:
  - 4 streams x 64 (vector length)
  - Uses as much extra memory as the Earth Simulator
- Unvectorized and unstreamed loop in “shifti” slows down the calculation to a crawl
  - 54% of the time spent in that routine according to “pat”
  - Was only 11% on the ES
The culprit in shifti

- "Unstreamed" and "unvectorized" loop due to nested if blocks:

```fortran
    do m=m0,mi
        zetaright=min(2.0*pi,zion(3,m))-zetamax
        zetaleft=zion(3,m)-zetamin
        if( zetaright*zetaleft > 0 )then
            zetaright=zetaright*0.5*pi_inv
            zetaright=zetaright-real(floor(zetaright))
            msend=msend+1
            kzi(msend)=m
            if( zetaright < 0.5 )then
                msendright(1)=msendright(1)+1
                iright(msendright(1))=m
            else
                msendleft(1)=msendleft(1)+1
                ileft(msendleft(1))=m
            endif
        endif
    enddo
```
New loop in shift

!dir$ preferstream
  do imm=1,4
!dir$ prefervector
  do m=(imm-1)*mi/4+1,imm*mi/4
    zetaright=min(2.0*pi,zion(3,m))-zetamax
    zetaleft=zetamin-zion(3,m)
    alpha=pi2*aint(1.0-pi4_inv*zetaleft)
    beta=pi2*aint(1.0-pi4_inv*zetaright)
    kappa=pi2*aint(1.0+zetaleft*zetaright*pi2sq_inv)
    aright=(alpha+zetaleft) - (beta+zetaright) - kappa
    aleft=(alpha+zetaleft) - (beta+zetaright) + kappa
    if( aright > 0.0 )then
      msend_r(imm)=msend_r(imm)+1
      kzi_r(msend_r(imm),imm)=m
    endif
    if( aleft < 0.0 )then
      msend_l(imm)=msend_l(imm)+1
      kzi_l(msend_l(imm),imm)=m
    endif
  enddo
endo
Did it work?

- Yes, the overall time spent in shifti went from 54% to only 4%!!

- On the Earth Simulator, the compiler can only deal with a single conditional statement within a loop in order to vectorize that loop. Solution: split the loop in 2 parts.

- Improved performance on the ES but not as dramatic as on the X1.
Memory used by the vectorized version of GTC (per processor)

- For micell=10 memory on the ES is up to 8 times more than one the Power 3!
- It gets better as the number of particles per cell increases
Flops/sec for higher particle resolution

64-Processor/MSP test runs

Number of particles per cell

Performance (Gflops)

X1
ES
P4
P3
High resolution simulations

• The algorithm to “push” the particles is very efficient on vector machines.
• This allows us to run high resolution simulations using a large number of particles.
• More particles means
  – more phase space resolution (velocity + configuration space)
  – lower discrete particle noise/fluctuations
  – longer simulations
  – higher efficiency on vector computers
• Without using OpenMP, we are back to only 1D domain decomposition and a max of 64-128 processors
• How to improve concurrency?
New parallel model: Domain decomposition + particle splitting

- **1D Domain decomposition:**
  - Several MPI processes can now sit in a section of the torus

- **Particle splitting method:**
  - The particles in a toroidal section are equally divided between several MPI processes

- Particles randomly distributed between processors within a toroidal domain.

- **No OpenMP**

- Pure MPI version
Pure MPI parallel model ideal for newest large scale computers

- New large scale computers such as Blue Gene/L and Cray XT3 allow only message passing for communication between processors.
- The MPI-only version of GTC has been very successful on those platforms.
- It achieved the highest performance of 7.2 TFlops on the Earth Simulator using 4,096 processors.
- Used over 16,000 processors on the Blue Gene/L computer at IBM Watson.
- Largest GTC production simulation recently carried out on 4,800 processors of the Cray XT3 at ORNL using 28 billion particles.
Latest benchmark: weak scaling study with fixed device size

Compute Power of the Gyrokinetic Toroidal Code
Number of particles (in million) moved 1 step in 1 second

Latest vector optimizations
Not tested on Earth Simulator

S. Ethier, PPPL, Nov. 2005
Conclusions

• Benchmarking and optimizing work never ends
• New platforms with more processors and different characteristics are continuously being developed.
• To cope with the changes while wanting to achieve top performance, codes must be flexible and developers must be willing to modify their codes.
• Secret to high performance (in my opinion…)
  – Data access (must feed the processor as fast as possible)
  – Fast data access = good data layout
  – Minimize communications
• Speed is not a substitute to “right answer”…