SILICON GRAPHICS | The Source of Innovation and Discovery™

Silicon Graphics, Inc.

A Shared Memory Programming Technique Applied to Computational Chemistry Programs

Presented by:

Roberto Gomperts

roberto@sgi.com

sgi®

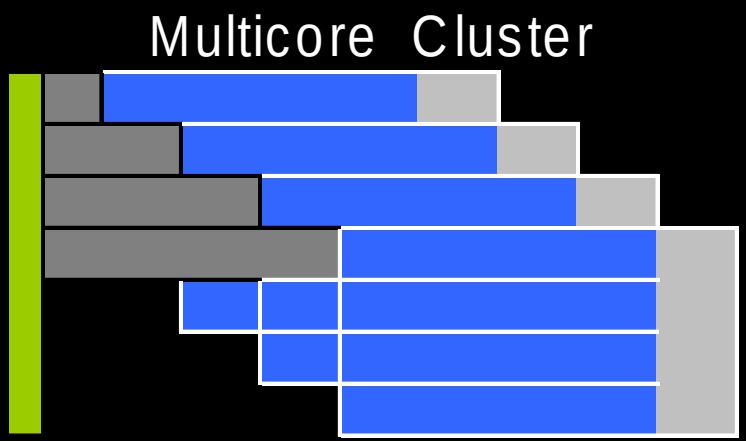
Have you heard…?

- “It is much easier to parallelize a code with OpenMP than with MPI”
  - “Automatic” parallelization
  - Just a directive here and there…
- “Parallelism with MPI is much more effective than with OpenMP”
  - Granularity of the approach

Topics

- Programming Model, Architecture
- Simplistic Execution profile for an MD kernel and an *ab-initio* kernel
- Simple approach to parallelize an MD kernel
- Simple approach to parallelize an *ab-initio* kernel
- Special issues: false sharing, dynamic load distribution
- Closing remarks

Architecture – Programming model: clusters

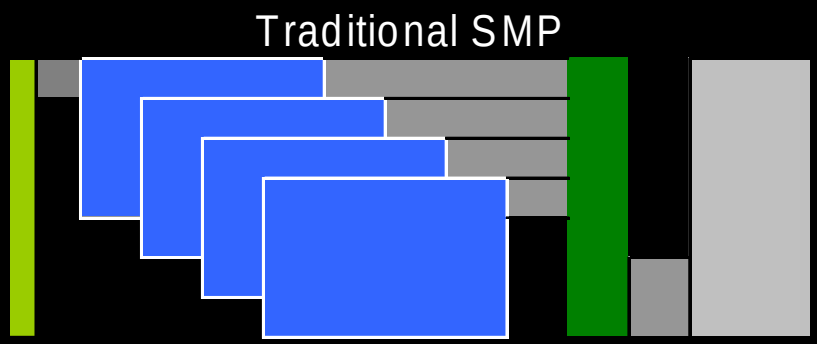


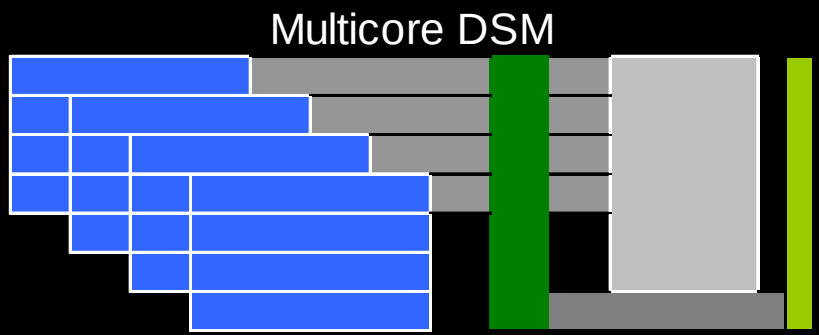
- Message passing to do B(:) = A(:)
  - Pack A(:)
  - Send A to CPU P
  - Receive A
  - Unpack A(:)
  - B(:) = A(:)

- Can Combine with shared memory programming model inside a box (hierarchical parallelism)

sgi

Architecture – Programming model: SHM



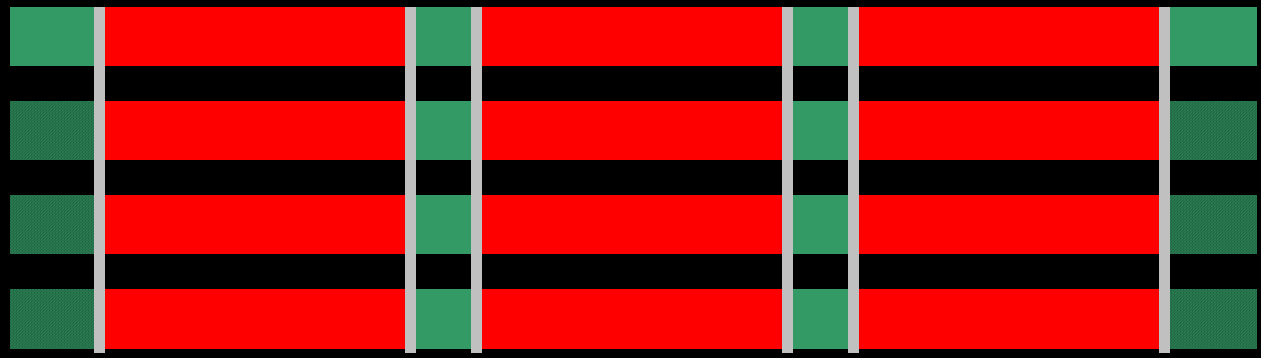


- Shared Memory programming model:
  - Insert directive
  - B(:) = A(:)
- Can also use Distributed Memory programming models
  - It is possible to take advantage of underlining memory architecture

sgi

Typical execution Model

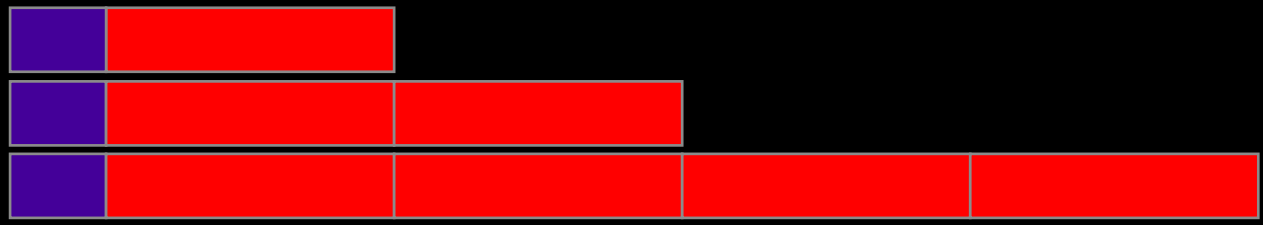
Distributed memory Model



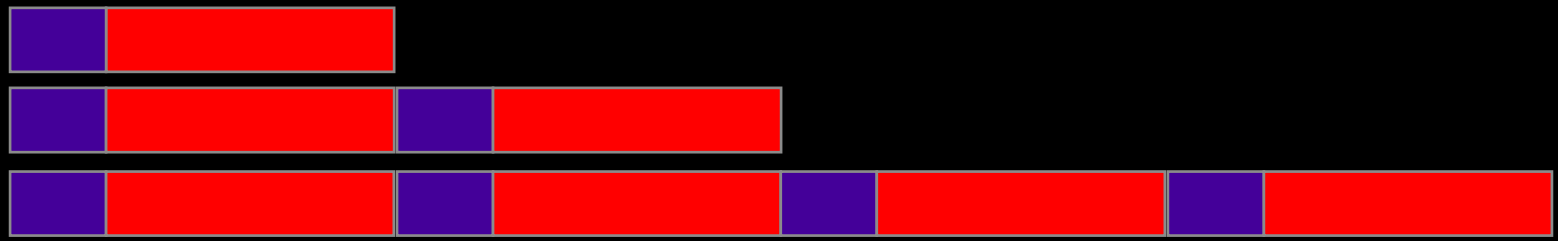
Shared memory Model

Typical Memory Usage

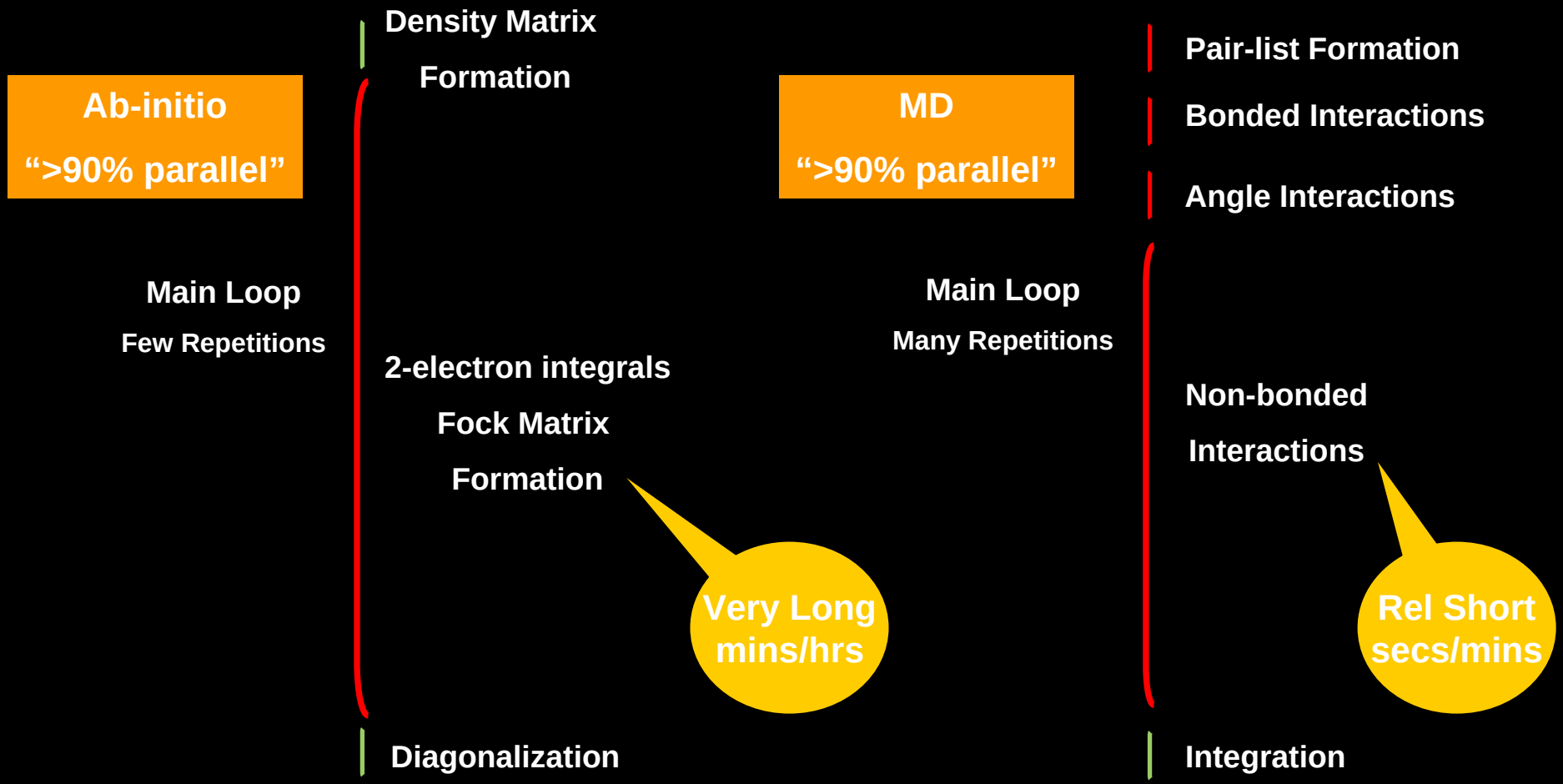
“Traditional” Shared memory



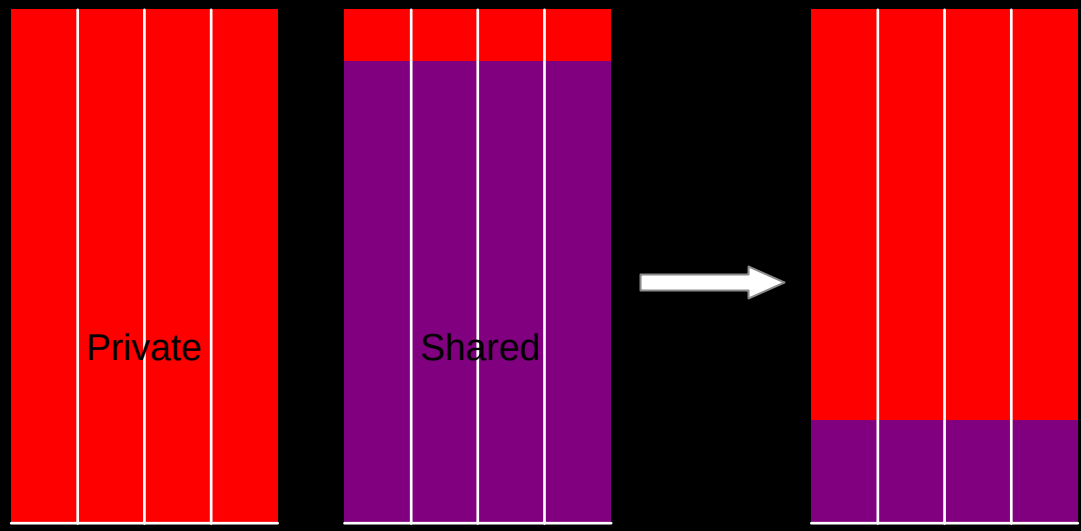
“Traditional” Distributed memory



Contrasting Parallel Behavior



The Basis of the Approach



Parallelization of an MD Program: Basic Algorithm (1)

```
Subroutine Energy
  en = 0; dxyz(:) = 0
  Call Bonds(en,dxyz)
  Call Angles(en,dxyz)
  Call NonBond(en,dxyz)
  …

Subroutine NonBond(en,dxyz)
  do i=1,nat
    dxyzj = 0
    do j=1,i-1
      exclude direct/far neighbors
      en = en + f(i,j,x,y,z,par)
      dxyzj = dxyzj + g(i,j,x,y,z,par)
    enddo
    dxyz(i) = dxyz(i) + h(i,x,y,z,dxyzj)
  enddo
  …
```

Parallelization of an MD Program (1)

```
Subroutine Energy
  en = 0; dxyz(:) = 0
  Call Bonds(en,dxyz)
  Call Angles(en,dxyz)
  Call NonBond(en,dxyz)
  …
```

```
Subroutine Energy_Driver
  allocate en_p(MAXPROC-1)
  allocate dxyz_p(:MAXPROC-1)
  en = 0; dxyz(:) = 0
C$OMP Parallel Do Private(ip,en_l)
  do ip=1,np
    if(iproc==np) then
      Call NonBond(en,dxyz,ip,np)
    else
      en_l = 0; dxyz_p(:,ip) = 0
      Call NonBond(en_l,dxyz_p(1,ip),ip,np)
      en_p(ip) = en_l
    endif
  enddo
  do ip=1,np-1
    Call Daxpy(3*nat,1.0d0,dxyz_p(1,ip),1,dxyz,1)
    en = en+en_p(ip)
  enddo
```

Parallelization of an MD Program (2)

```
Subroutine NonBond(en,dxyz)
  do i=1,nat
    dxyzj = 0
    do j=1,i-1
      exclude direct/far neighbors
      en = en + f(i,j,x,y,z,par)
      dxyzj = dxyzj + g(i,j,x,y,z,par)
    enddo
    dxyz(i) = dxyz(i) + h(i,x,y,z,dxyzj)
  enddo
```

```
Subroutine NonBond(en,dxyz,ip,np)
  do i=1,nat (distribute over np)
    dxyzj = 0
    do j=1,i-1
      exclude direct/far neighbors
      en = en + f(i,j,x,y,z,par)
      dxyzj = dxyzj + g(i,j,x,y,z,par)
    enddo
    dxyz(i) = dxyz(i) + h(i,x,y,z,dxyzj)
  enddo
```

Parallelization of an *ab-initio* Program: Basic Algorithm (1)

```
Subroutine SCFdriver
  allocate scratch(LARGE)
  fock(:,:) = 0
  do while(.not.converged)
    Call DirSCF(LARGE,scratch,fock)
    Call Eigen(fock)
  enddo
  …
```

```
Subroutine CalcInt(LARGE,scratch)
  do while(FitInLarge)
    do i,j,k,l
      scratch(i,j,k,l) = f(i,j,k,l)
    enddo
  enddo
…
```

```
Subroutine DirSCF(LARGE,scratch,fock)
  do while(.not. All_ijkl)
    Call CalcInt(LARGE,scratch)
    Call InFock(LARGE,scratch,fock)
  enddo
…
```

```
Subroutine InFock(LARGE,scratch,fock)
  do i=1,LARGE
    fock(ij,kl) = fock(ij,kl)+
      c(ij,kl)*scratch(i,j,k,l)
  enddo
…
```

Parallelization of an *ab-initio* Program: Option 1 (1)

```
Subroutine SCFdriver
  allocate scratch(LARGE)
  fock(:,:) = 0
  do while(.not. converged)
    Call DirSCF(LARGE,scratch,fock)
    Call Eigen(fock)
  enddo
  …
```

```
Subroutine SCFdriver
  allocate scratch(LARGE)
  allocate scr_p(LARGE,MAXPROC-1)
  allocate fock_p(:,:,MAXPROC-1)
  fock(:,:) = 0
  do while(.not. converged)
C$OMP Parallel DO Private(ip)
    do ip=1,np
      if(ip==np) then
        Call DirSCF(LARGE,scratch,fock,ip,np)
      else
        fock(:,:,ip) = 0
        Call DirSCF(LARGE,scr_p(1,ip),
             fock_p(1,1,ip),ip,np)
      endif
    enddo
    do ip=1,np-1
        Call Daxpy(ij*kl,1.0d0,fock_p(1,1,ip),1,
            fock,1)
    enddo
    Call Eigen(fock)
  enddo
```

sgi

Parallelization of an *ab-initio* Program: Option 1 (2)

```
Subroutine DirSCF(LARGE,scratch,fock)
  do while(.not. All_ijkl)
    Call CalcInt(LARGE,scratch)
    Call InFock(LARGE,scratch,fock)
  enddo

Subroutine CalcInt(LARGE,scratch)
  do while(.FitInLarge)
    do i,j,k,l
      scratch(i,j,k,l) = f(i,j,k,l)
    enddo
  enddo

Subroutine InFock(LARGE,scratch,fock)
  do i=1,LARGE
    fock(ij,kl) = fock(ij,kl) +
                  c(ij,kl)*scratch(i,j,k,l)
  enddo
```

```
Subroutine DirSCF(LARGE,scratch,fock,ip,np)
  do while(.not. All_ijkl)
    Call CalcInt(LARGE,scratch,ip,np)
    Call InFock(LARGE,scratch,fock)
  enddo

Subroutine CalcInt(LARGE,scratch,ip,np)
  do while(.FitInLarge)
    do i,j,k,l (get unique ijkl batches)
      scratch(i,j,k,l) = f(i,j,k,l)
    enddo
  enddo

Subroutine InFock(LARGE,scratch,fock)
  do i=1,LARGE
    fock(ij,kl) = fock(ij,kl) +
                  c(ij,kl)*scratch(i,j,k,l)
  enddo
```

sgi

Parallelization of an *ab-initio* Program: Option 2 (1)

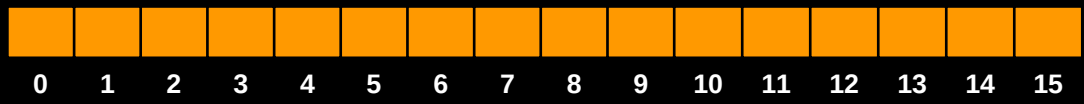
```
Subroutine SCFdriver
  allocate scratch(LARGE)
  fock(:,:) = 0
  do while(.not.converged)
    Call DirSCF(LARGE,scratch,fock)
    Call Eigen(fock)
  enddo
  …
```

```
Subroutine SCFdriver
  allocate scratch(LARGE)
  allocate fock_p(:,MAXPROC-1)
  fock(:,:) = 0
  sl = LARGE/np; lst_sl = LARGE – sl*(np-1)
  do while(.not.converged)
C$OMP Parallel DO Private(ip)
    do ip=1,np
      if(ip==np) then
        Call DirSCF(lst_sl,
          scratch(1+sl*(ip-1)),fock,ip,np)
      else
        fock(:,:,ip) = 0
        Call DirSCF(LARGE,scratch(1,sl*(ip-1)),
             fock_p(1,1,ip),ip,np)
      endif
    enddo
    do ip=1,np-1
        Call Daxpy(ij*kl,1.0d0,fock_p(1,1,ip),1,
            fock,1)
    enddo
    Call Eigen(fock)
  enddo
```

sgi

Miscellaneous issues

- False sharing
  - Cache line size (Itanium 128 bytes)



- Dynamic load distribution
  - Too fine: possible contention on critical region
  - Too coarse: possible lack of load balancing

Closing remarks

- Shown an effective methodology to apply coarse grain parallelism with OpenMP
- This will become increasingly important as node sizes grow (multi-core chips)
- Apply the right tool for the right problem:
  - Distributed Memory parallelism for inter-node sections
  - Shared memory parallelism for intra-node sections