Leveraging Transparent Data Distribution in OpenMP via User-level Dynamic Page Migration

Dimitrios S. Nikolopoulos[1], Theodore S. Papatheodorou[1], Constantine D. Polychronopoulos[2], Jesús Labarta[3] and Eduard Ayguadé[3]

[1]LHPCA (U. Patras), [2]CSRL (UIUC), [3]CEPBA (UPC)

OpenMP

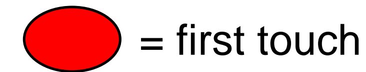
- **Simplicity**
  - incremental development of parallel programs
  - directive-based programming paradigm
- **Portability**
  - independence from architecture
  - independence from OS
- **De facto standard for**
  - SMP, ccNUMA, software DSM
- **Performance frequently inferior to message-passing: sensitivity to page placement**

Why pages are not where I would like?

```
integer, parameter :: n=10002
real*8,  parameter :: tol=1.0d-10
real*8             :: X(n,n), Y(n,n), err

read *, ((X(i,j), i=1,n), j=1,n)
do iter=1,20000

!$omp parallel do private(i)
  do j=2,n-1
    do i=2,n-1
       Y(i,j) = (X(i-1,j) + X(i+1,j) + X(i,j-1) + X(i,j+1)) / 4.0
    end do
  end do
  err = 0.0d0
!$omp parallel do private(i), reduction(+:err)
  do j=2,n-1
    do i=2,n-1
       X(i,j) = (Y(i-1,j) + Y(i+1,j) + Y(i,j-1) + Y(i,j+1)) / 4.0
       err = err + (X(i,j)-Y(i,j))**2
    end do
  end do
  if (err < tol) exit
end do
end
```



Paths of an L2 cache miss

node

node

node

node

IN

P P

DSMc

M

node

node

node

Origin2000≈400ns-900ns
WildFire ≈1700ns

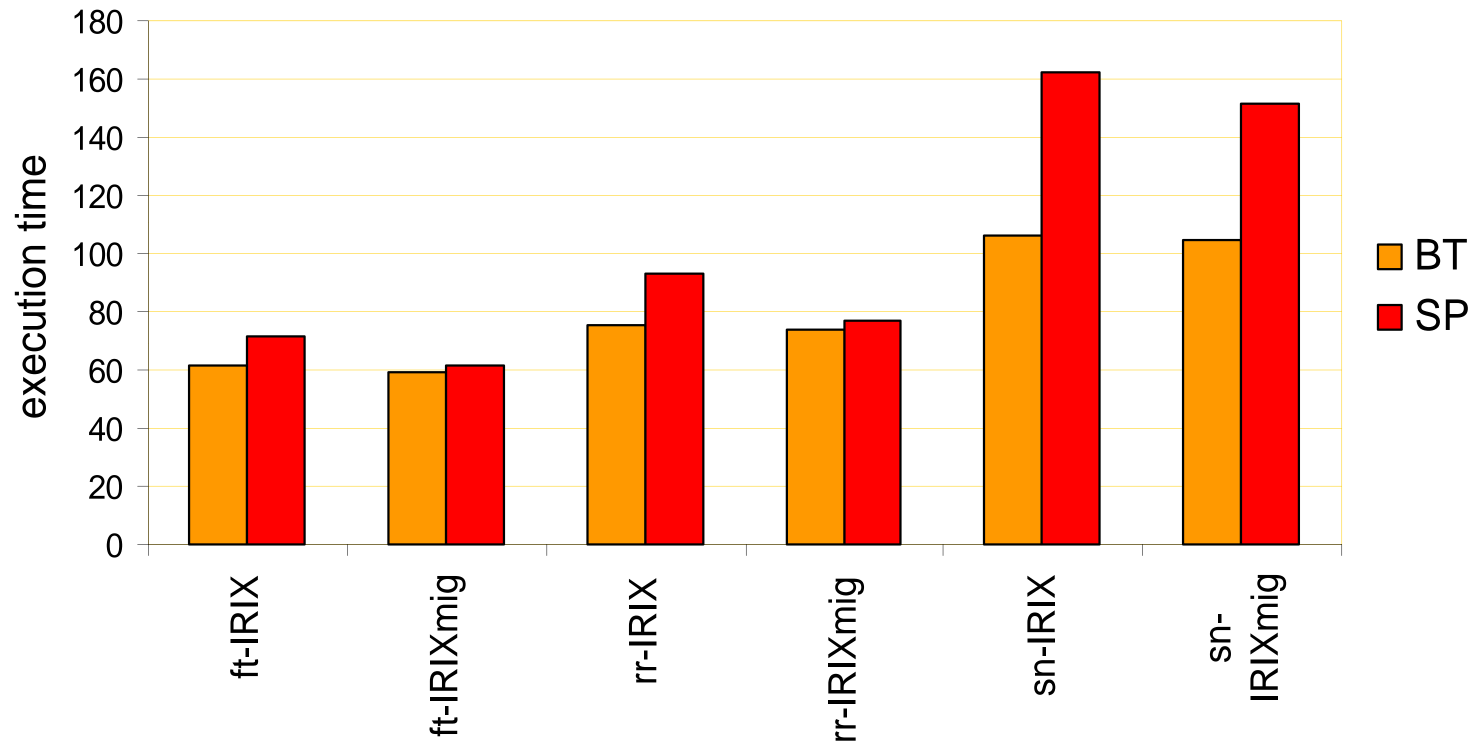
Origin2000≈300ns
WildFire ≈250ns

CEPBA nanos

OS dynamic page migration

- **First-touch page placement**
  - the processor that experiences the first TLB miss on a page maps it locally
  - reasonable amounts of locality for several programs, simple and easy to implement
  - cannot handle dynamic reference patterns, thread migrations, thread preemptions
- **Dynamic page migration based on page reference counters**
  - count the references issued by each node to a page in memory
  - migrate the page based on a criterion of competitiveness

OS dynamic page migration



Explicit management of locality

- **User-directed data layout:**
  - !dec$ omp distribute var_name(block)
  - data placement: page or element granularity
  - !dec$ omp numa: to control computation placement

```
      real :: a(n,n)
!dec$ distribute (*,cyclic) :: a(n,n)

      do k=1,n-1
        do m = k+1, n
          a(m,k) = a(m,k) / a(k,k)
        end do
!dec$ omp numa
!$omp parallel do private(i)
        do j = k+1, n
          do i = k+1, n
            a(i,j) = a(i,j) - a(i,k) *a(k,j)
          end do
        end do
      end do
```

Summary and motivation

- **Explicit data distribution and redistribution**
  - Effective: accurate, customisable to the semantics of the application
  - Compromises simplicity
  - Hard to standardize: not supported by OpenMP

- **OS support for NUMA-aware memory management**
  - Locality-aware page placement
  - Dynamic page migration and replication
  - Transparent to the programmer
  - Questionable performance: inflexible, inaccurate, poor responsiveness

A different approach

- **Try to mimic data mapping directives by using compiler-driven dynamic page migration**
  - transparency
  - no source code and OpenMP API modifications
- **What do we need?**
  - track hardware counters attached to memory pages
  - dynamically optimize data placement at runtime
  - compiler-inserted migrations
- **Detect thread migrations in a multiprogrammed environment** (not covered in this paper)

Experimental setting

- **User-level implementation on IRIX 6.5**
- **SGI Origin2000**
- **OpenMP implementations of the NAS benchmarks**
  - first-touch page placement
  - customized to the O2000 memory hierarchy
- **Emulation of alternative page placement schemes:**
  - round-robin
  - worst-case (single node)

User-level dynamic page migration

OpenMP Compiler

Hot areas identification

Instrumentation

reference counters

UPMlib core
Migration algorithms

scheduling info

IRIX

page placement

migration

/proc interface

IRIX mmci

Kernel Scheduler

Approximating data distribution

- **Identification of hot memory areas from the compiler**
- **Where to insert migration actions? early identification of the optimal home node for each page:**
  - iterative codes: at the end of each iteration
  - non-iterative codes: periodic sampling of hardware counters
- **Orthogonal to the initial page placement scheme**

Approximating data distribution

```
...
do step = 1, niter
   call compute_rhs ()
   call x_solve ()
   call y_solve ()
   call z_solve ()
   call add

enddo
...
```

```
subroutine x_solve
...
!$omp parallel do
  do k=1,grid_points(3)-2
    do j=1,grid_points(2)-2
       do i=0,isize
       ...
    enddo
  enddo
...
```

Approximating data distribution

```
...
call upmlib_init()
call upmlib_memrefcnt(u,size)
call upmlib_memrefcnt(rhs,size)
call upmlib_memrefcnt(forcing,size)
do step = 1, niter
   call compute_rhs ()
   call x_solve ()
   call y_solve ()
   call z_solve ()
   call add
   call upmlib_migrate_memory ()
enddo
...
```

Approximating data distribution

NAS BT and SP (class A, 32 processors)

execution time

180
160
140
120
100
80
60
40
20
0

ft-IRIX
ft-IRIXmig
ft-upmlib
rr-IRIX
rr-IRIXmig
rr-upmlib
sn-IRIX
sn-IRIXmig
sn-upmlib

BT
SP

Histogram of remote references

NAS BT, array u

remote reference

70000
60000
50000
40000
30000
20000
10000
0

20 40 60 80 100 120 140 160 180 200

iteration number

Advantages

- **Accuracy**
  - Page migration based on accurate snapshots of the complete memory reference pattern
- **Timeliness**
  - Places all pages early in the optimal home node
- **Effective substitute to an initial data distribution scheme**
- **Transparent to the programmer**
- **Orthogonal to the OS page placement scheme**

Periodic page migration mechanism

- **Invoked periodically**
- **Selection of sampling frequency**
  - adapts to the duration of phase changes
- **Selection of sampling rate**
  - Limits the overhead of sampling to a small fraction of the sampling period
- **Effectiveness depends of**
  - Duration of phases
  - Computational granularity

Approximating data redistribution

- **Identify phase transition points**
  - Sequence of !$omp parallel do/sections constructs with similar data access patterns
- **Recording phase**
  - snapshots of hardware counters at phase transition points in distinct software buffers
  - pair-wise comparisons of consecutive snapshots
  - identify optimal home node for each page for each phase in isolation
- **Replay phase**
  - move pages at phase transitions

Approximating data redistribution

```
...
do step = 1, niter
   call compute_rhs ()
   call x_solve ()
   call y_solve ()
   call z_solve ()
   call add

enddo
...
```

```
subroutine x_solve
...
!$omp parallel do
  do k=1,grid_points(3)-2
    do j=1,grid_points(2)-2
      do i=1,isize
      ...
    enddo
  enddo
...
```

Approximating data redistribution

```
...
do step = 1, niter
   call compute_rhs ()
   call x_solve ()
   call y_solve ()
   call z_solve ()
   call add

enddo
...
```

```
subroutine x_solve
```

```
subroutine y_solve
...
!$omp parallel do
  do k=1,grid_points(3)-2
    do i=1,grid_points(1)-2
      do j=0,jsize
      ...
    enddo
  enddo
...
```

Approximating data redistribution

```
...
do step = 1, niter
   call compute_rhs ()
   call x_solve ()
   call y_solve ()
   call z_solve ()
   call add

enddo
...
```

```
subroutine x_solve
```

```
subroutine y_solve
```

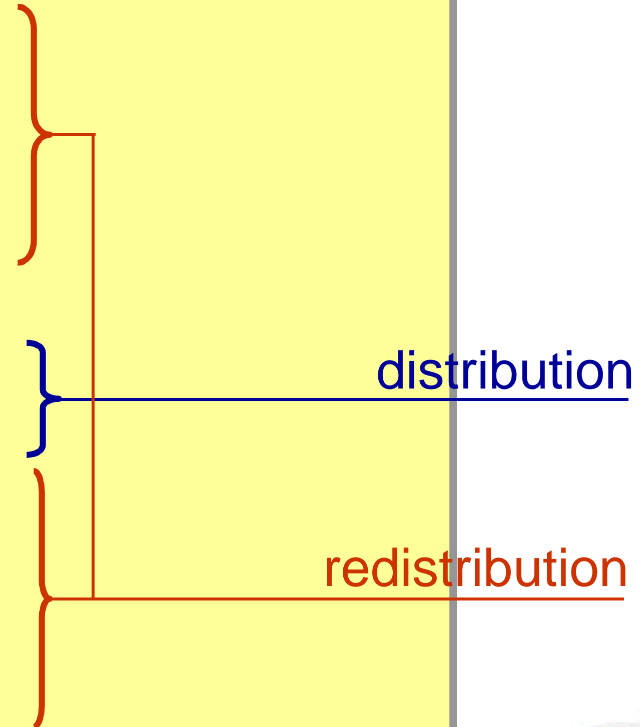
```
subroutine z_solve
...
!$omp parallel do
  do j=1,grid_points(2)-2
    do i=1,grid_points(1)-2
      do k=0,ksize
        ...
    enddo
  enddo
...
```

Approximating data redistribution

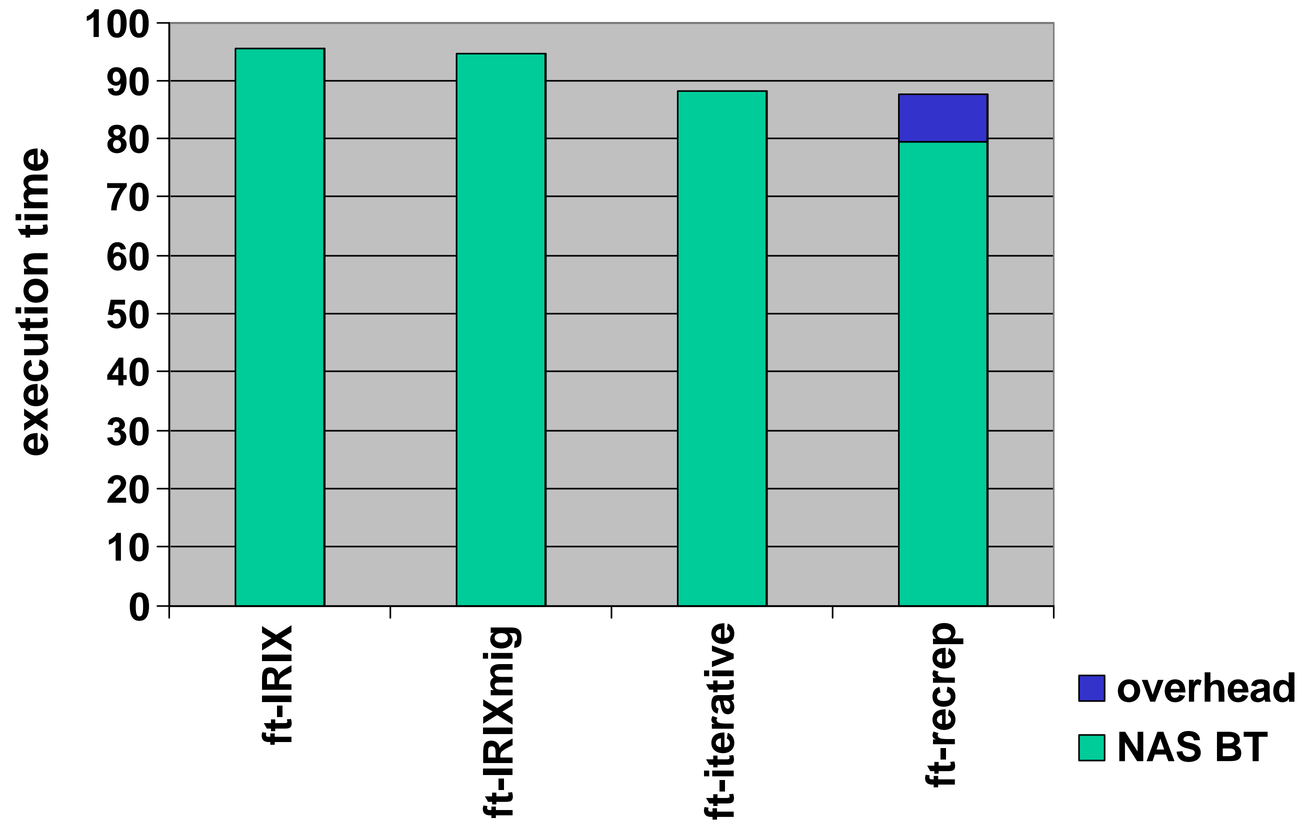
```
...
do step = 1, niter
   call compute_rhs ()
   call x_solve ()
   call y_solve ()
   if (step .eq. 1) then
     call upm_lib_record ()
   else
     call upmlib_reply ()
   endif
   call z_solve ()
   if (step .eq. 1) then
     call upm_lib_record ()
   else
     call upmlib_undo ()
   endif
   call add
enddo
...
```

Data distribution and redistribution

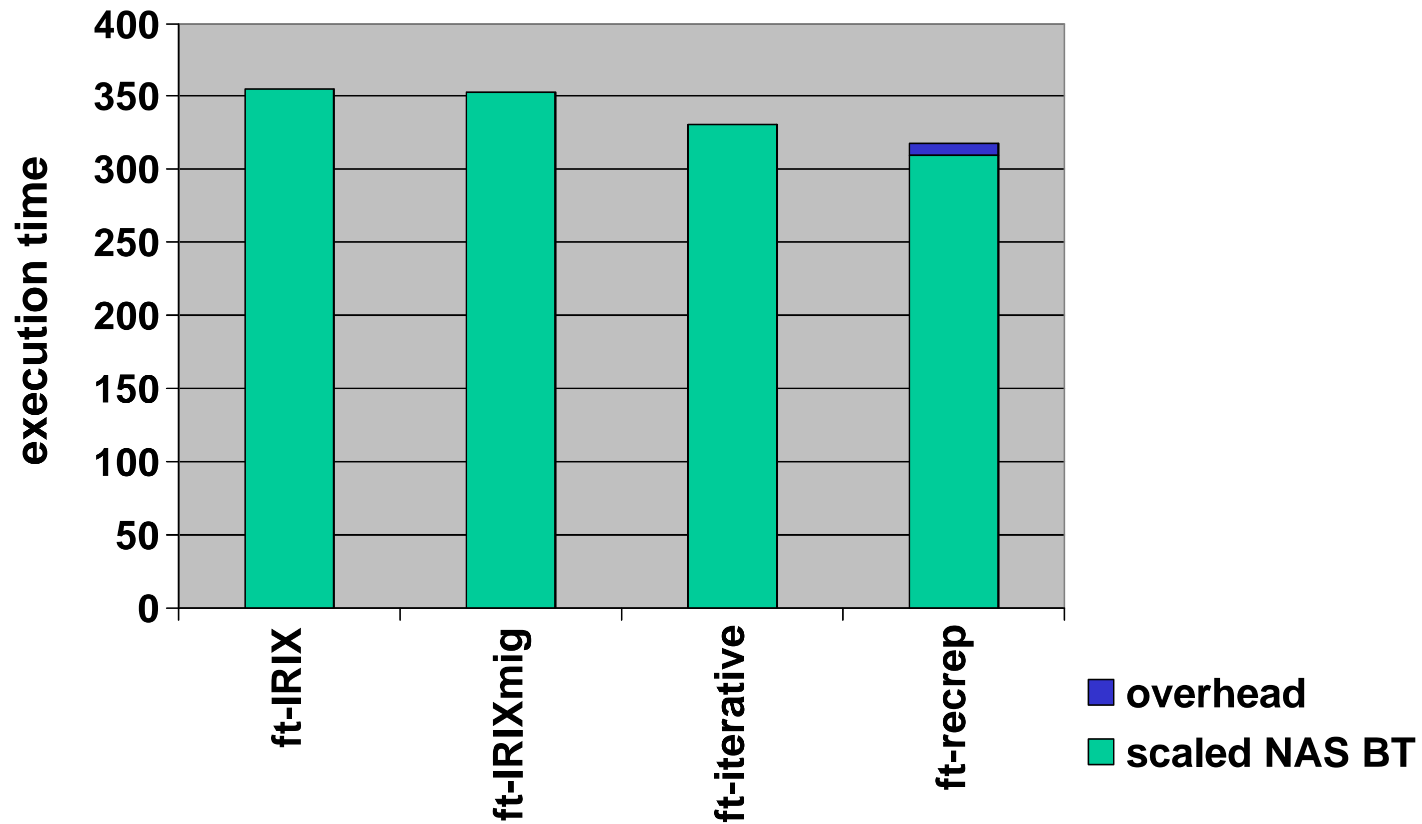
```
...
do step = 1, niter
   call compute_rhs ()
   call x_solve ()
   call y_solve ()
   if (step .eq. 10) then
     call upm_lib_record ()
   else if (step .gt. 10) then
     call upmlib_reply ()
   endif
   call z_solve ()
   if (step .lt. 10) then
     call upm_migrate_memory ()
   else if (step .eq. 10) then
    call upm_lib_record ()
   else
     call upmlib_undo ()
   endif
   call add
enddo
...
```



Performance of record/reply



Performance of record/reply



Conclusions

- **User-level page migration engine: substitute for data distribution/redistribution in OpenMP**
  - Transparency
  - No modification of source code and OpenMP API
- **Highly effective in place of static data distribution**
- **Sensitive to the granularity of phases when used in place of data redistribution**
- **Implementation status:**
  - UPMlib already implemented
  - compiler in progress within the OpenMP NanosCompiler (multilevel parallelism with thread groups extension)

papers / release of NthLib, NanosCompiler

Visit our booth at SC2000