Automatic Generation of OpenMP Directives and Its Application to Computational Fluid Dynamics Codes

WOMPEI
October 17, 2000

Haoqiang Jin, Michael Frumkin
Computer Sciences Corporation
Jerry Yan
NAS Systems Division
NASA Ames Research Center

Outline

- Motivation
- Parallelization Tools
- Generating OpenMP Directives
- Results from CFD Applications
- Future Work

Code Parallelization

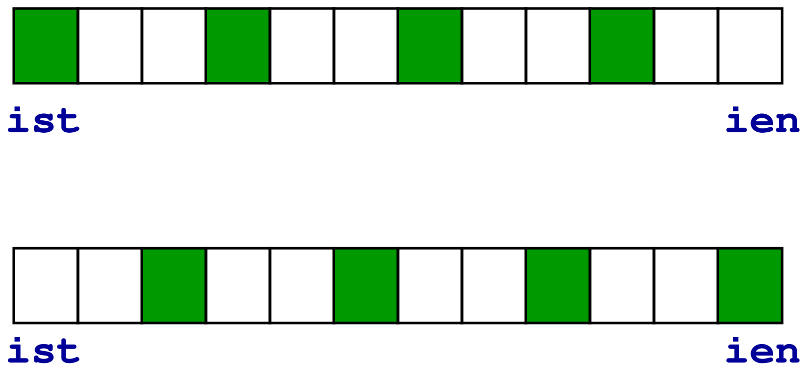
- **By hand**
  - time-consuming
  - error-prone
  - but, better performance
- **Automated compiler**
  - limited depth of analysis (time constraint)
  - limited performance
- **OpenMP directives**
  - less intrusive than message passing to original code
  - easier to insert manually but still error prone!

Nature of the Tools for Parallelization

- **User expectation**
  - fully automatic
  - high parallel efficiency
  - efficient memory usage
- **Reality**
  - interaction essential due to implicit information on input variables
  - user optimization usually needed for high performance
  - constraints on "depth" of analysis acceptable to user
- **Narrowing the gap**
  - automate much of the tedious work
    - accurate data dependence analysis
  - allow user to focus on small part of the problem
    - effective user interface
  - reduce user time and error

Importance of User Knowledge

```
   read*,a,ist,ien,istep
S1 do j=1,nj
      do i=ist,ien,istep
S2      a(i)=...
      enddo
      ...
      do i=ien,ist,-istep
S3       ...=a(i)
      enddo
   enddo
```



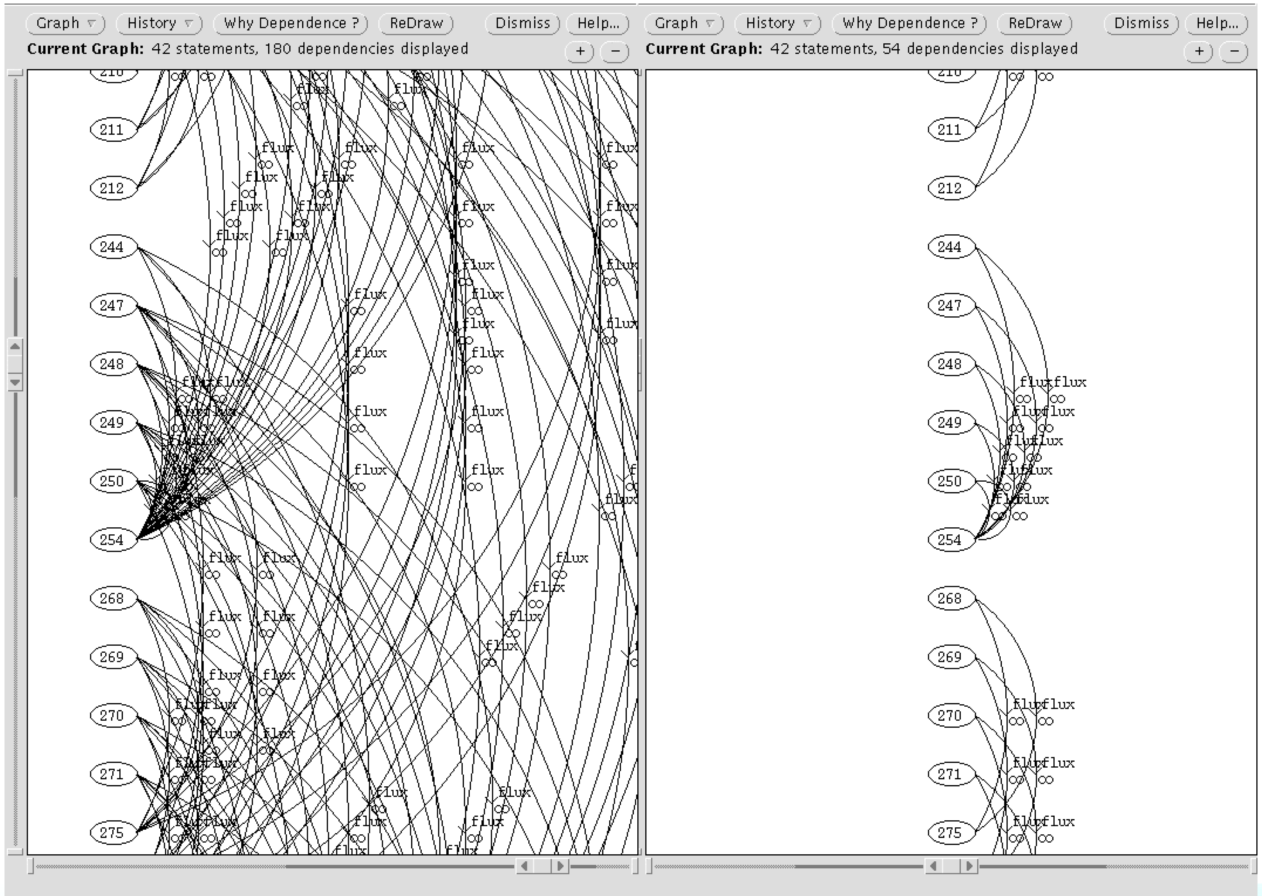
- If it is not CERTAIN that **ien – ist** divisible by **istep** then data used in S_3 may originate from earlier iteration of S_1 or from READ
- Without user knowledge to assist the analysis a dependence is assumed.
- May lead to serialization or communication.

Parallelization Tools

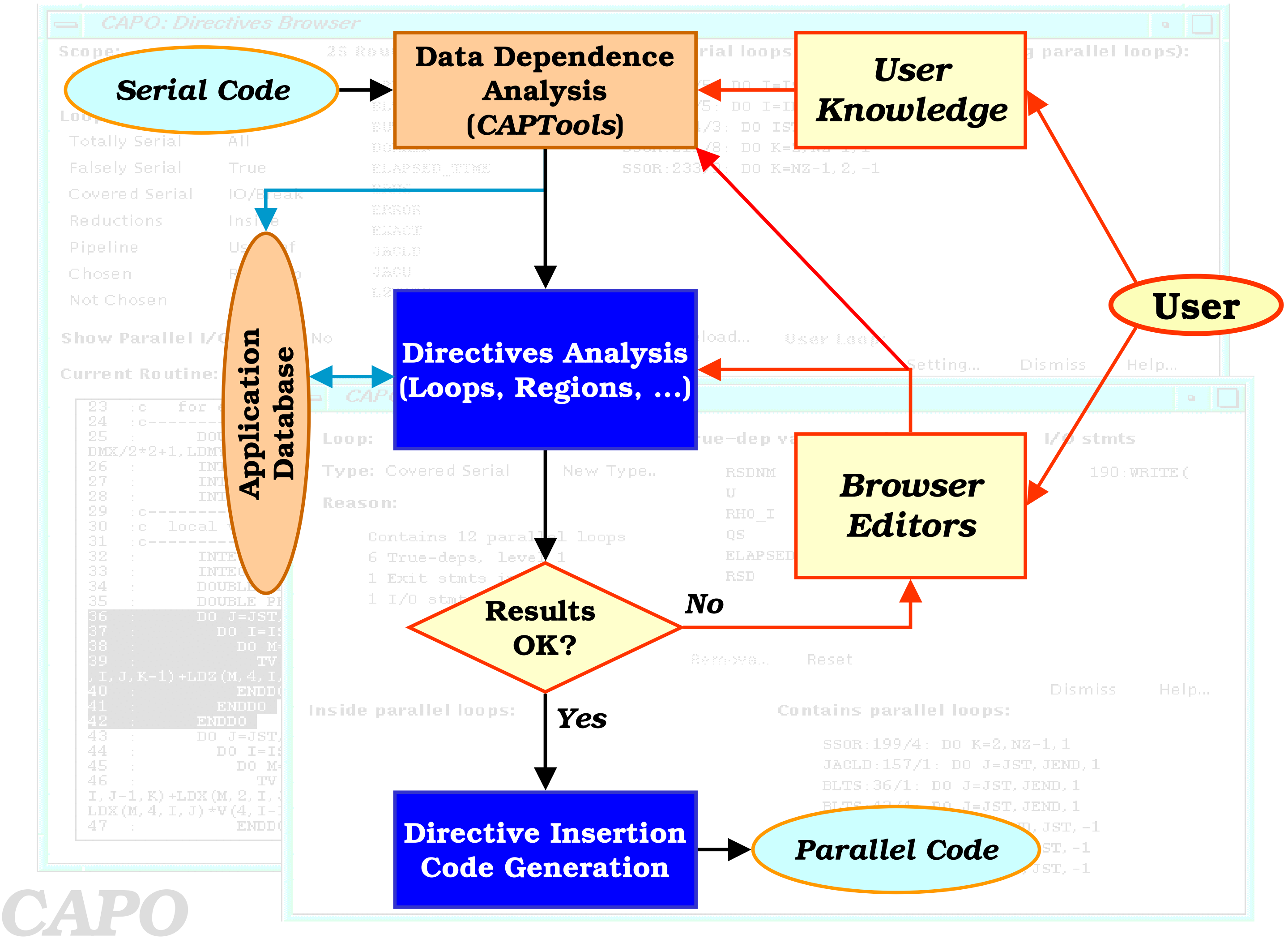
- **CAPTools**
  - interactive semi-automatic toolkit
  - developed at the University of Greenwich
- **CAPTools main feature**
  - accurate and in-depth data dependence analysis
  - domain decomposition strategy for parallelization
  - generic set of message-passing calls, mapped to MPI/PVM
  - a number of browsers for user to interact with the parallelization process
- **CAPO**
  - generating OpenMP directives
  - based on CAPTools (dependence analysis)

Quality Dependence Analysis

- Example for NAS-PAR benchmark code LU
- Standard analysis graph (left) and CAPTools analysis (right)
- Parallelization at best poor with standard graph



Generating OpenMP Parallel Code



Automatic Generation of OpenMP Code

- **Identify parallel loops**
- **Identify and define SHARED and PRIVATE variables**
- **Generate software pipeline constructs using OpenMP where this is necessary e.g. NPB LU code**
- **Identify and define REDUCTION operations**
- **Detect and produce THREADPRIVATE directives for common blocks**

Complexities in OpenMP Code Generation

- Identify and define PARALLEL regions using quality interprocedural dependence analysis to identify parallelism as high as possible
- Merge consecutive parallel regions into a single region to reduce overheads in thread startup and synchronization at end of parallel region
- Detect and produce NOWAIT directives when this is allowed
- Perform automatic routine duplication when a routine usage conflict occurs at different calling points

Private Variables in Common blocks

- **When THREADPRIVATE directive is not possible**
- **Variables are automatically added to argument lists in routines**

Original code

```
   common /csub/b(im),a(im,jm)
   do 10 j=1,ny
     call sub(j,nx)
     do 10 i=1,nx
       a(i,j)=b(i)
10 continue

   subroutine sub(j,nx)
   common /csub/c(im),a(im,jm)
   do 20 i=1,nx
     c(I)=a(i,j)…
20 continue
   end
```

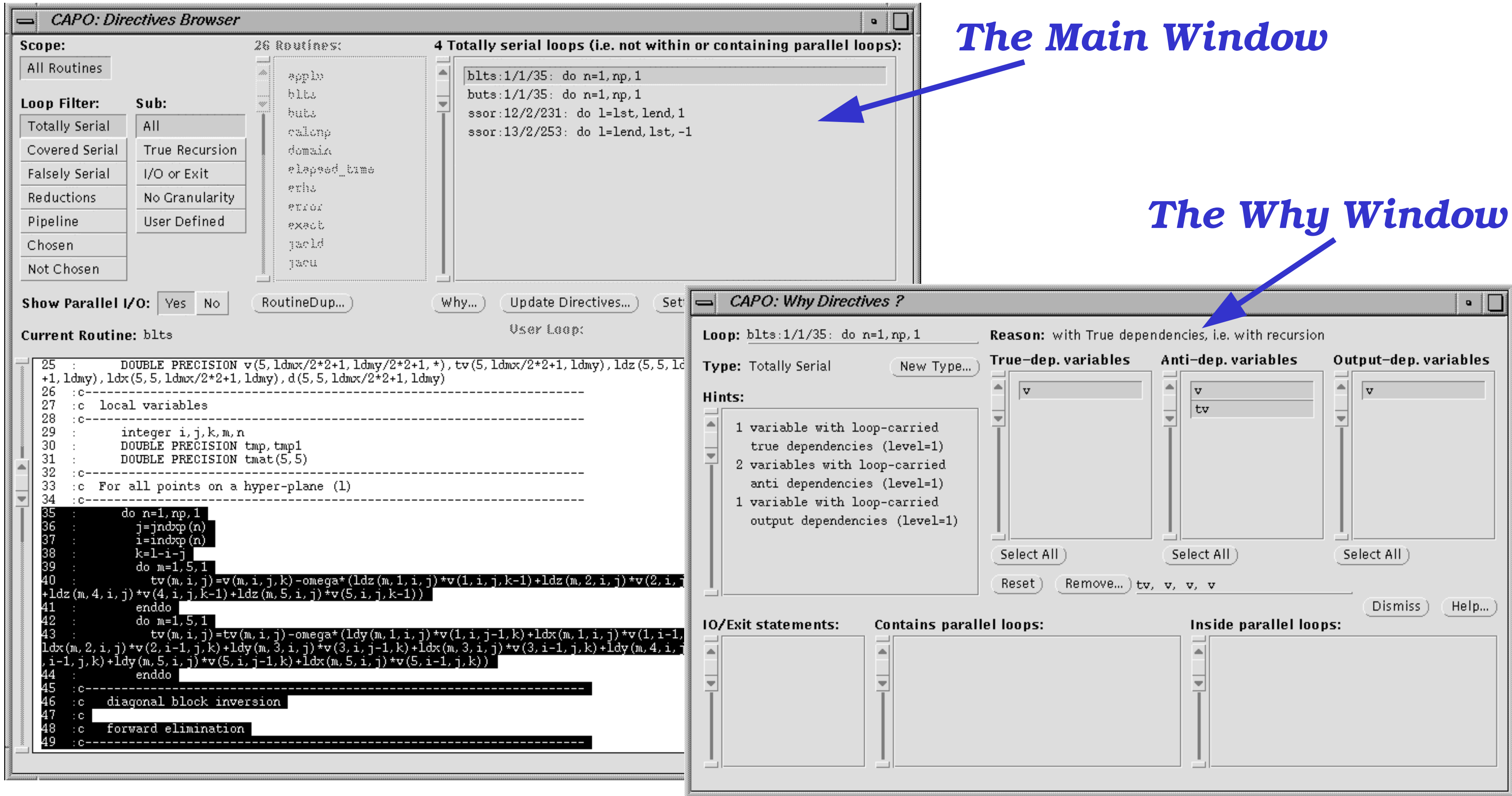
Parallel code

```
   common /csub/b(im),a(im,jm)
!$OMP PARALLEL DO PRIVATE(i,j,B)
   do 10 j=1,ny
     call sub(j,nx,B)
     do 10 i=1,nx
       a(i,j)=b(i)
10 continue
!$OMP END PARALLEL DO

   subroutine sub(j,nx,C)
   common /csub/c_cap(im),a(im,jm)
   dimension C(im)
   do 20 i=1,nx
     c(i)=a(i,j)…
20 continue
   end
```

Presenting Loop Level Information — The Directives Browser

- **Focus on the *problematic* code sections**
- **Understand why a loop is serial or parallel**
- **Prune the analysis (tool follows the conservative path)**



Loop Types Identified with the Browser

- **Totally Serial (Potentially Severe)**
  - Loop carried true dependence present.
  - Not contained in, or containing ANY parallel loops - entirely serial
  - Sequential execution can prevent effective parallel performance.
- **Covered Serial (May be important)**
  - Also a serial loop, but contains or is contained in a parallel loop so some parallelism will be exploited.
  - If containing parallel loops, parallel performance can be enhanced by parallelism at this higher level.
- **Falsely Serial (May be important)**
  - Does not contain loop carried true dependence -should be able to exploit parallelism
  - Serial due to loop carried pseudo (memory re-use) dependence by a non-privatizable variable

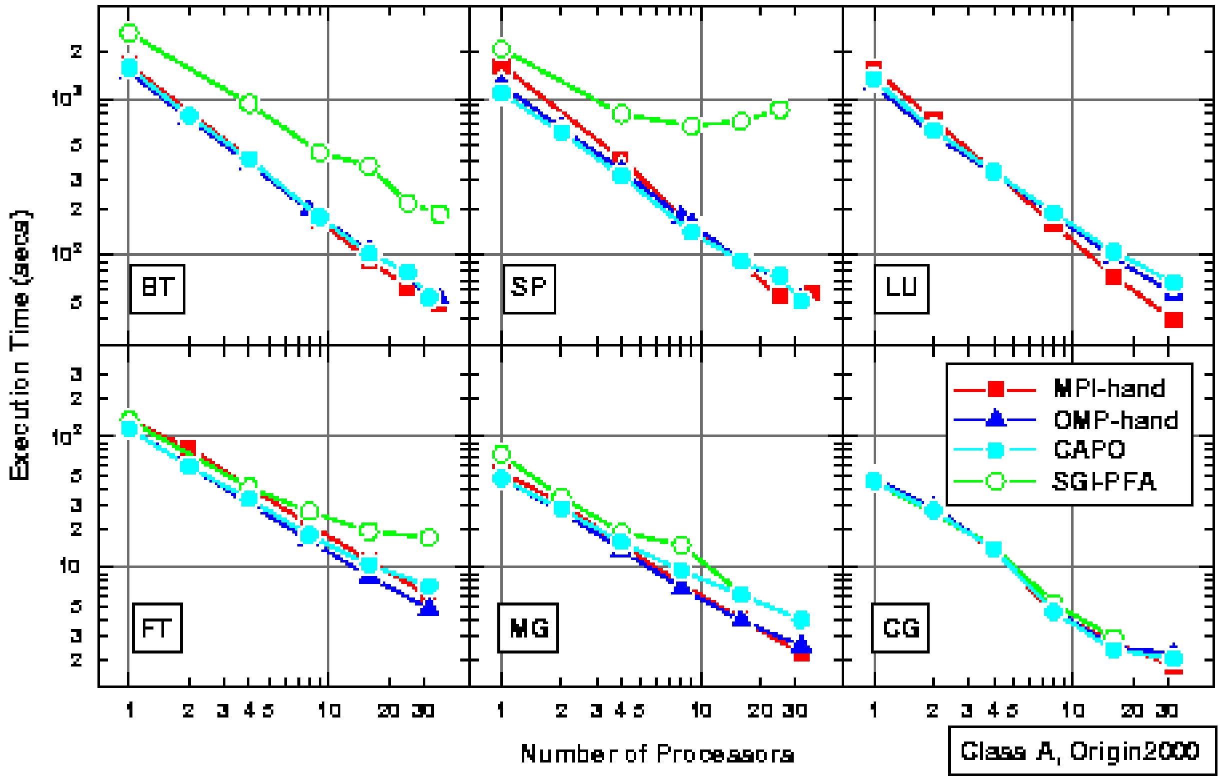
More Loop Types

- **Reduction**
  - Loop contains a recognized reduction operation.
- **Pipeline**
  - Loop identified as a software pipeline.
- **Chosen**
  - Parallel loop that is not nested within other parallel loops.
  - Current Loop level at which parallel DO directive is inserted.
- **Not Chosen**
  - Parallel loop not chosen due to the selection of other parallel loops from the categories above.

Parallelization of CFD Applications

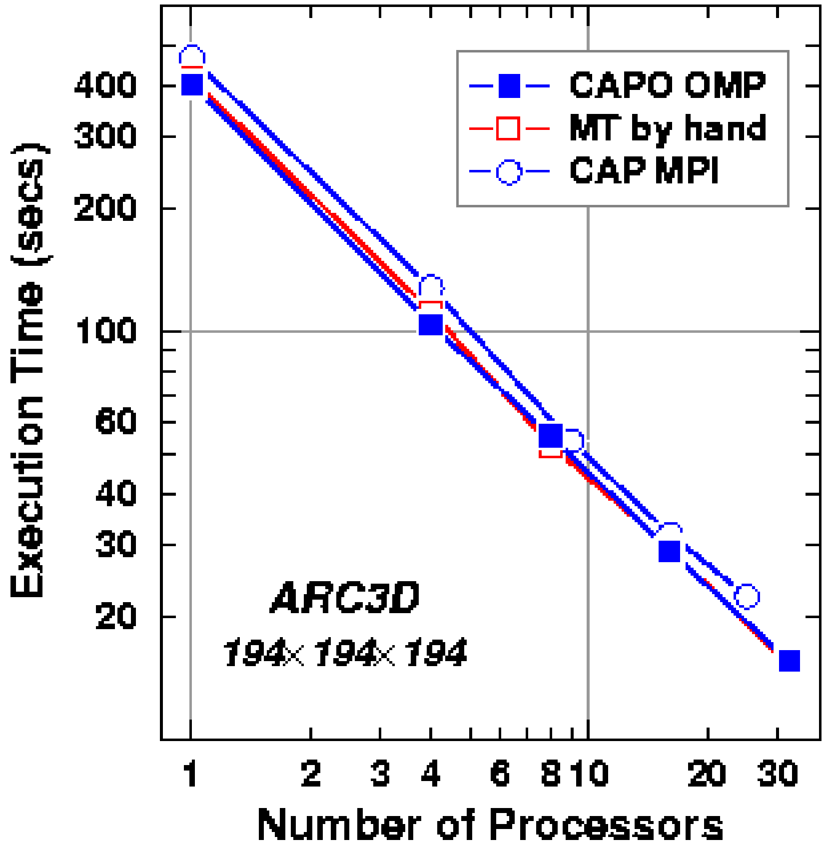
- **NAS Parallel Benchmarks**
  - six serial versions (BT, SP, LU, FT, CG, MG)
- **Three CFD codes**
  - ARC3D, OVERFLOW, INS3D
- **Parallelization**
  - strategy: outer-most loops for large granularity
  - straightforward for NPBs and ARC3D
  - some user interaction for OVERFLOW and INS3D
- **Testing Platform**
  - SGI Origin2000, MIPSpro Fortran 77 compiler

NAS Parallel Benchmarks



on SGI Origin2000, 195MHz

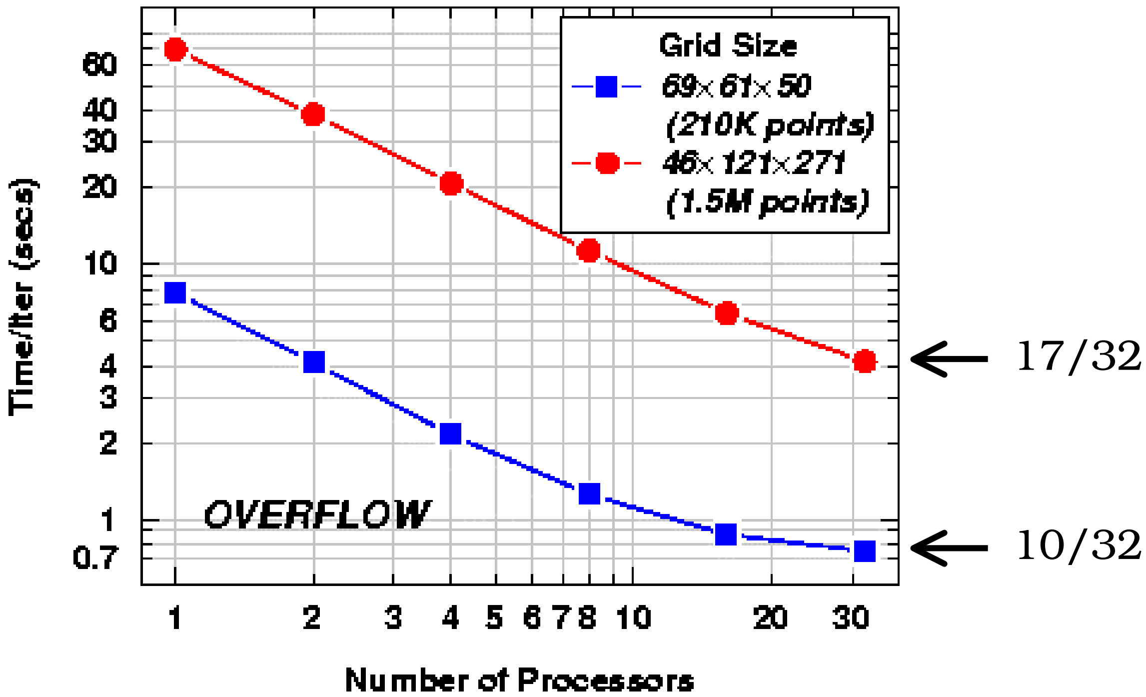
ARC3D



on SGI Origin2000, 195MHz

OVERFLOW Performance

- Two test cases: 210K and 1.5M points
- Single zone



on SGI Origin2000, 195MHz

User Interaction (OVERFLOW)

- **Removing *false* dependences**
  - example of *false* dependence

```
      NTMP2 = JD*KD*31
      DO 100 L=LS,LE
        CALL GETARX(NTMP2,TMP2,ITMP2)
        CALL WORK(L,TMP2(ITMP2,1),TMP2(ITMP2,7),...)
        CALL FREARX(NTMP2,TMP2,ITMP2)
100   CONTINUE
```

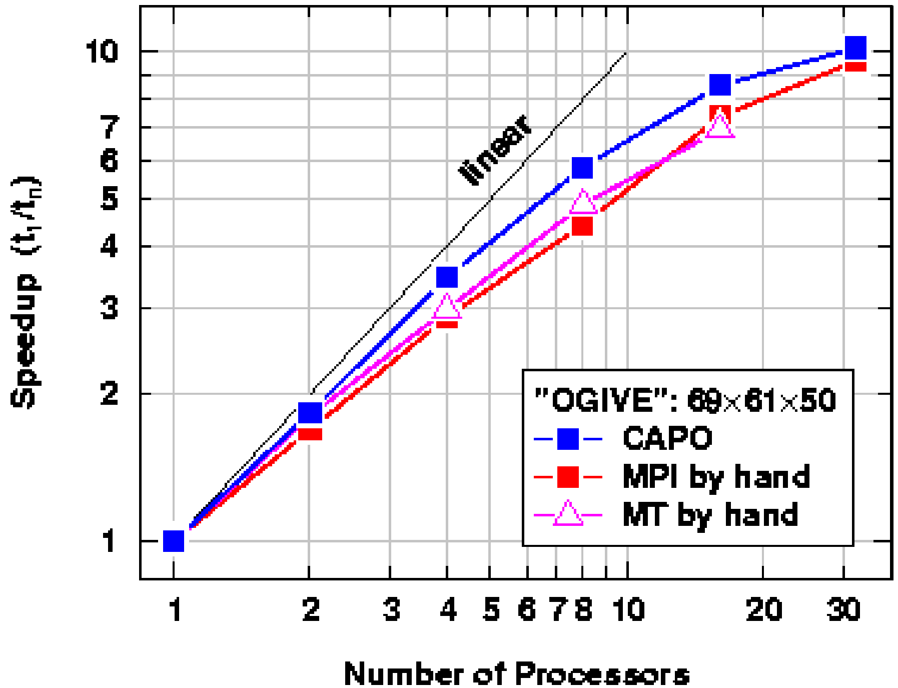
Allocate TMP2

Free TMP2

  - **GETARX** and **FREARX** calls for dynamic memory allocation
  - loop-carried dependences caused by **TMP2** were removed

Performance Comparison

- Three versions of OVERFLOW
  - ⇒ OpenMP directives from CAPO
  - ⇒ MPI by hand
  - ⇒ Multi-Tasking directives by hand



Summary of CAPO on CFD Applications

Application	*BT,SP,LU*	*FT,CG,MG*	*ARC3D*	*OVERFLOW*	*INS3D*
Code Size	~3000 lines benchmark	~2000 lines benchmark	~4000 lines	851 routines 100K lines	256 routines 41K lines
Dependence Analysis	30 mins to 1 hour	10 mins to 30 mins	40 mins	25 hours	42 hours
Code Generation	< 5 mins	< 5 mins	< 5 mins	1 day	2 days
Testing	1 day	1 day	1 day	3 days	3 days
Performance Compared to Hand-coded Version	within 5-10%	within 10% for CG 30-36% for FT,MG	within 6%	slightly better	no hand-coded paral-lel version

- **Dependence Analysis**:
  on Indy R5K (NPBs and ARC3D) and Sun E10K (OVERFLOW and INS3D)
- **Code Generation:**
  including user interaction with the tool and code restructure
- **Testing:**
  debugging and running the parallel codes

Future Work

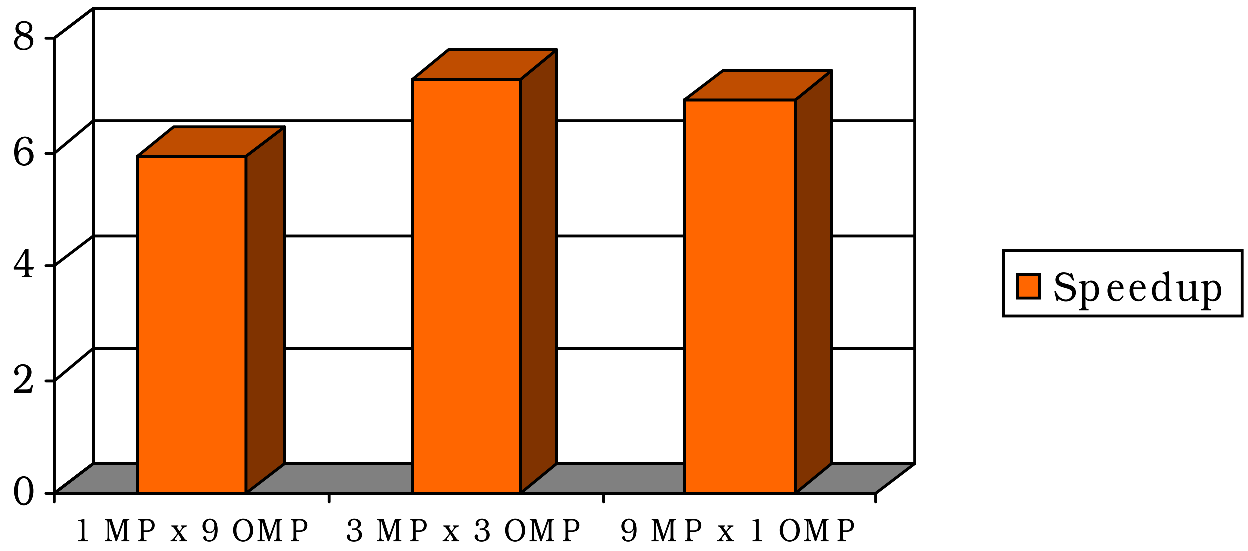
- **Enhance performance**
  - automated code transformation for cache performance
  - automated data placement
- **Combine MP and OMP for multi-level parallelism**
  - CAPTools/CAPO
    - message passing parallel code based on an SPMD paradigm
    - OpenMP parallel code based on a loop distribution approach
  - why not combine the two strategies:

```
do k=1,nk           Parallel loop exploited with MPI
  do j=1,nj         Parallel loop exploited with OpenMP
    do i=1,ni
      ...
    end do
  end do
end do
```

  - first pass: message passing, second pass: OpenMP

The Hybrid Approach

- Tested on BT benchmark
- CAPTools generated message passing code followed by
- CAPO generated OpenMP code
- Test on 195 MHz Origin2000, 1 vs 9 CPUs



Contact Information

- **CAPO**
  `mailto:capo@nas.nasa.gov`
  `http://www.nas.nasa.gov/Tools/CAPO/`

- **CAPTools**
  `http://captools.gre.ac.uk/`