Advanced Parallelizing Compiler (APC) Overview

APC Technology Group

Evaluation Method

- **Select Standard benchmarks**
  1. **SPEC CFP2000:Most familiar Scientific & Engineering Benchmark Suite**
  2. **SPEC CFP95**
  3. **NAS-PB 2.3-Serial**
- **Only FORTRAN77 programs!**
- **Use the SPEC CFP2000_base compiler option with vender specific automatic parallelizing options.**
- **Use the same compiler option for backend code generation.**

Result

- **IBM RS/6000**
  - The APC compiler shows ***2.5*** times faster speedup than the original.
- **IBM pSeries690**
  - The APC compiler shows ***3.5*** times faster speedup than the original.
- **HP AlphaServer GS160**
  - The APC compiler shows ***2.1*** times faster speedup than the original.
- **SGI Origin2000**
  - The APC compiler shows ***2.7*** times faster speedup than the original.

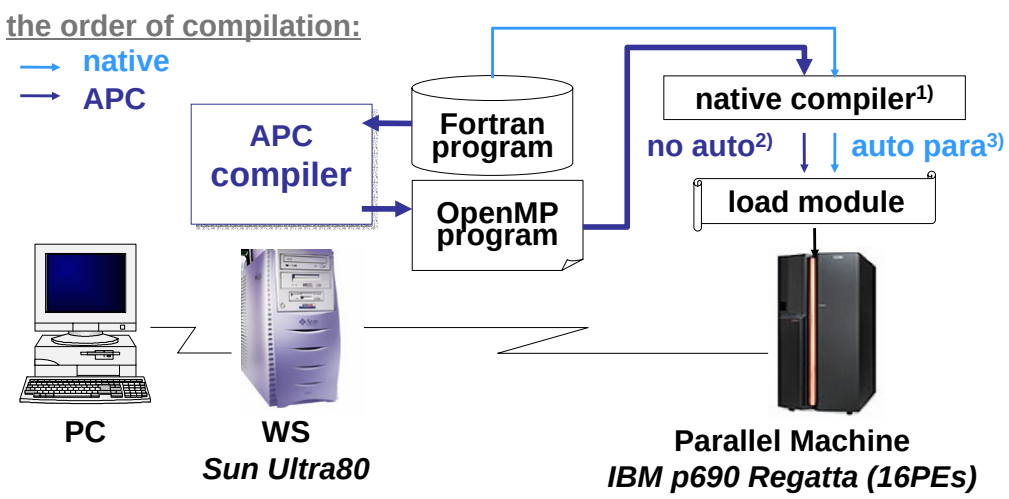
Conclusion

- **Automatic multi-grain parallelization is effective for many benchmarks.**

  On IBM pSeries690, the APC compiler attains 3.5 times faster than the original compiler.
- **On every machine, the APC compiler attains more than double performance compared with the original compilers.**

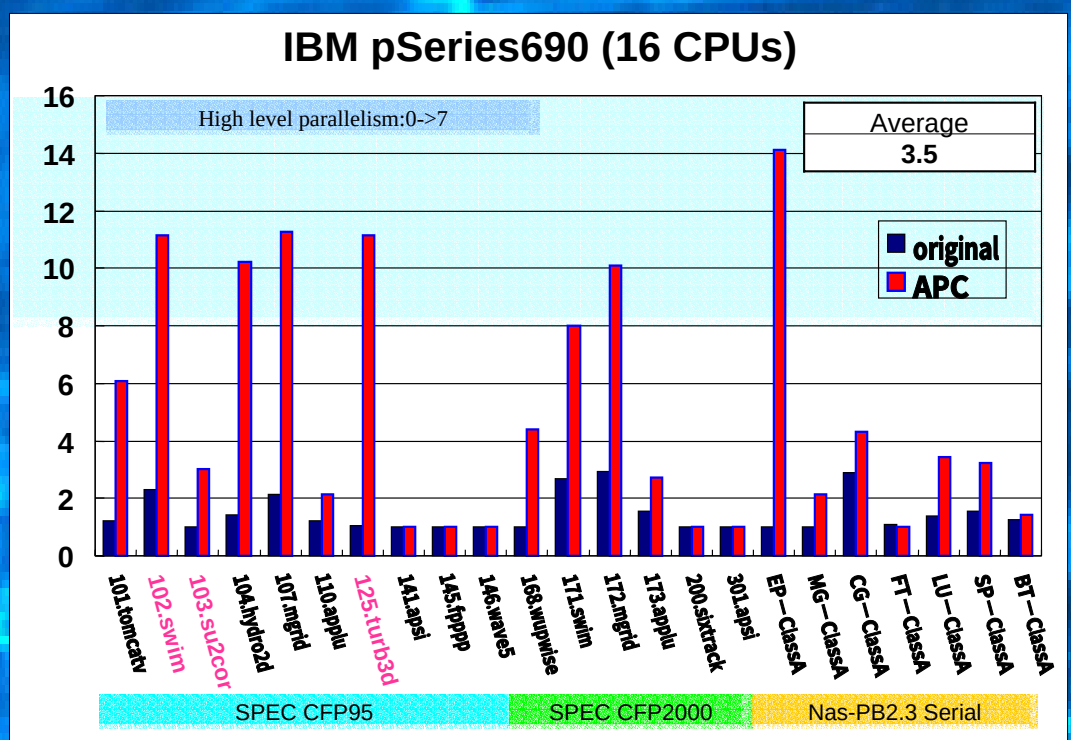
Demonstration

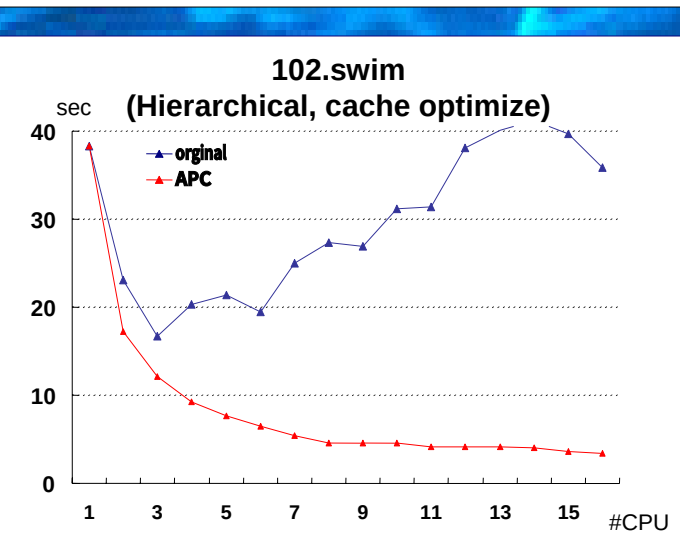
We will run several benchmark programs (swim, su2cor, and turb3d) on the IBM's latest high-end multiprocessor pSeries690 (16way). That makes you convince the performance advantage of the APC compiler.

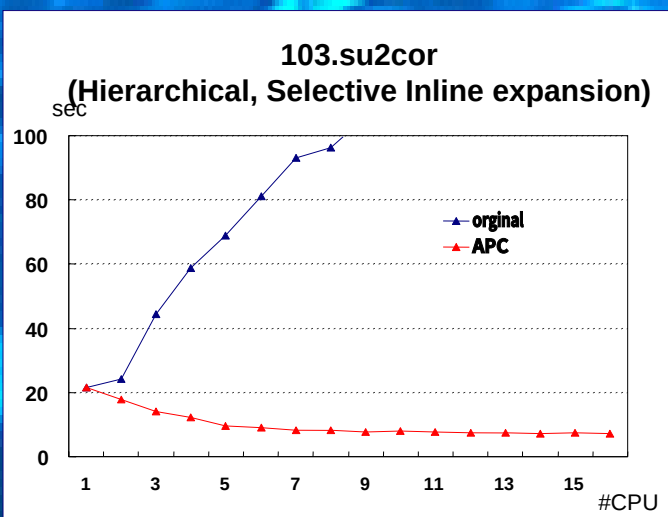


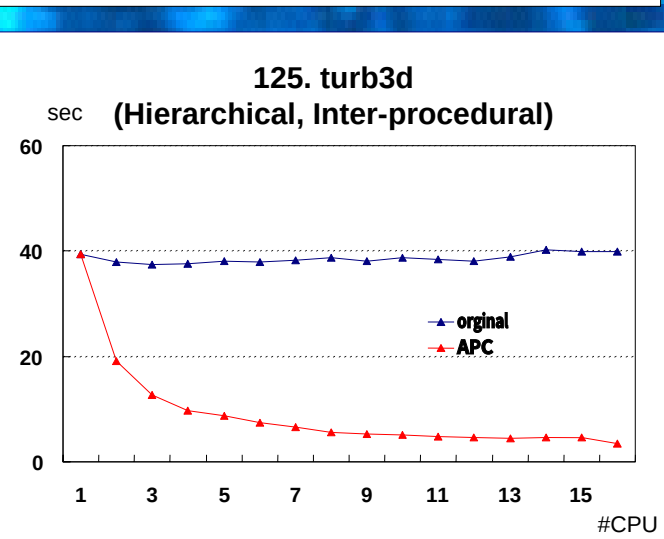
1) XL Fortran for AIX Ver. 8.1 2) -qsmp=noauto or –qsmp=omp 3) -qsmp=auto

Evaluation System



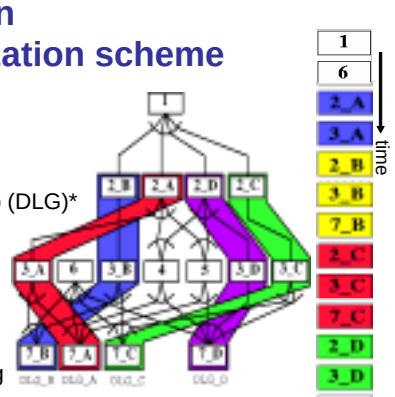






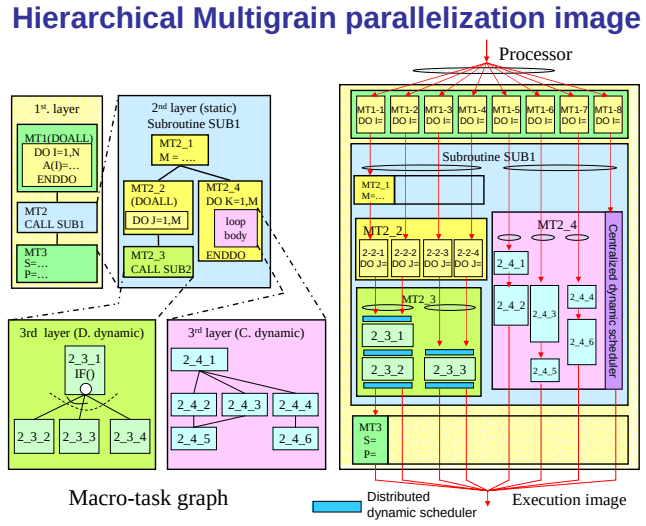
Cache optimization using Data Localization scheme

- Divide the loops (2, 3, and 7) considering cache size
- Make Data Localizable Group (DLG)*
  - * group of macro -tasks to be assigned to the same thread for passing the shared data through cache
- Apply Partial Static scheduling for data localization



Scheduling result on single processor

Hierarchical Multigrain parallelization image



Interprocedural Optimization Cloning and Constant Propagation

Original program

```
do J = 1, 64
  call ZFFT(J, 1, 1, 1)
  call ZFFT(J, 1, 2, 1)
  call ZFFT(J, 1, 3, 1)
  call ZFFT(J, 1, 4, 1)
  call ZFFT(J, 1, 5, 1)
  call ZFFT(J, 1, 6, 1)
enddo
```

```
ZFFT(.., IND, IS)
IF (IS = -1) THEN
  IF (IND = 1) THEN
    call DCFT(...)
  ENDIF
  IF (IND = 2) THEN
    call DCFT(...)
  ENDIF
  ...
```

```
DCFT(..)
call CFFT(..)
  ...
call CFFT(..)
...
```

```
CFFT(..)
call FFTZ1(..)
  ...
call FFTZ2(..)
  ...
```

After applying Cloning and Constant Propagation

```
!$OMP PARALLEL DO
do J = 1, 64
  call Cln11ZFFT(J, 1, 1, 1)
  call Cln10ZFFT(J, 1, 2, 1)
...
enddo
!$OMP END PARALLEL DO
```

```
Cln11ZFFT(.., IND, IS)
call Clon3DCFT(...)
```

```
Cln3DCFT(..)
call Cln3CFFT(..)
  ...
call Cln4CFFT(..)
...
```

```
Cln3CFFT(..)
do
  ...
enddo
```