Lecture 12

Parallelization

I Basic Parallelization

II Data dependence analysis

III Interprocedural parallelization

Chapter 11.1-11.1.4

Parallelization of Numerical Applications

- **DoAll loop parallelism**
  - Find loops whose iterations are independent
  - Number of iterations typically scales with the problem
  - Usually much larger than the number of processors in a machine
  - Divide up iterations across machines

Basic Parallelism

- **Examples**

```
FOR i = 1 to 100
   A[i] = B[i] + C[i]

FOR i = 11 TO 20
   a[i] = a[i-1] + 3

FOR i = 11 TO 20
   a[i] = a[i-10] + 3
```

- **Does there exist a data dependence edge between two different iterations?**
- **A data dependence edge is loop-carried if it crosses iteration boundaries**
- **DoAll loops: Loops without loop-carried dependences**

Recall: Data Dependences

- **True dependence**

```
a     =
      = a
```

- **Anti-dependence**

```
      = a
a     =
```

- **Output dependence**

```
a     =
a     =
```

Affine Array Accesses

- **Common patterns of data accesses: (i, j, k are loop indexes)**

```
A[i], A[j], A[i-1], A[0], A[i+j], A[2*i], A[2*i+1]
A[i,j], A[i-1, j+1]
```

- **Array indexes are affine expressions of surrounding loop indexes**
  - Loop indexes i_n, i_{n-1}, ... , i_1
  - Integer constants c_n, c_{n-1}, ... , c_0
  - Array index: $c_n i_n + c_{n-1} i_{n-1} + ... + c_1 i_1 + c_0$
  - Affine expression: linear expression + a constant term (c_0)

II. Formulating Data Dependence Analysis

```
FOR i := 2 to 5 do
  A[i-2] = A[i]+1;
```

- **Between read access `A[i]` and write access `A[i-2]` there is a dependence if**
  - there exist two iterations i_r and i_w within the loop bounds, s.t.
  - iterations i_r & i_w read & write the same array element,respectively

$$\exists integers \qquad i_w, i_r \qquad 2 \le i_w, i_r \le 5 \qquad i_r = i_w - 2$$

- **Between write access `A[i-2]` and write access `A[i-2]` there is a dependence if**

$$\exists integers \qquad i_w, i_v \qquad 2 \le i_w, i_v \le 5 \qquad i_w - 2 = i_v - 2$$

  - To rule out the case when the same instance depends on itself: add constraint $i_w \neq i_v$

Memory Disambiguation

is

Undecidable at Compile Time

```
read(n)
For i =
  a[i] = a[n]
```

Domain of Data Dependence Analysis

- **Only use loop bounds and array indexes that are affine functions of loop variables.**

```
for i = 1 to n
   for j = 2i to 100
      a[i+2j+3][4i+2j][i*i] = ...
         ... = a[1][2i+1][j]
```

- **Assume a data dependence between the read & write operation if**
  - there exist a read instance with indexes i_r, j_r & a write instance with indexes i_w, j_w

$$\exists integers \qquad i_r, j_r, i_w, j_w \qquad 1 \le i_w, i_r \le n \qquad 2i_w \le j_w \le 100$$

$$2i_r \le j_r \le 100$$

$$i_w + 2j_w + 3 = 1 \qquad 4i_w + 2j_w = 2i_r + 1$$

- **Equate each dimension of array access; ignore non-affine ones**
  - No solution => No data dependence
  - Solution => there may be a depedence

Complexity of Data Dependence Analysis

- **Equivalent to integer linear programming**

$$\exists integer \quad \vec{i} \quad A_1\vec{i} \le \vec{b}_1 \quad A_2\vec{i} = \vec{b}_2$$

- **Integer linear programing is NP-complete**
  - O(size of the coefficients) or O(n^n)

Data Dependence

- **Many simple, identical tests**
  - most of them can be resolved very simply
  - memoization eliminates repeated computation
  - can be solved exactly in practice
- **omega package, Bill Pugh, University of Maryland**

Relaxing Dependences

- **Privatization**
  - Scalar

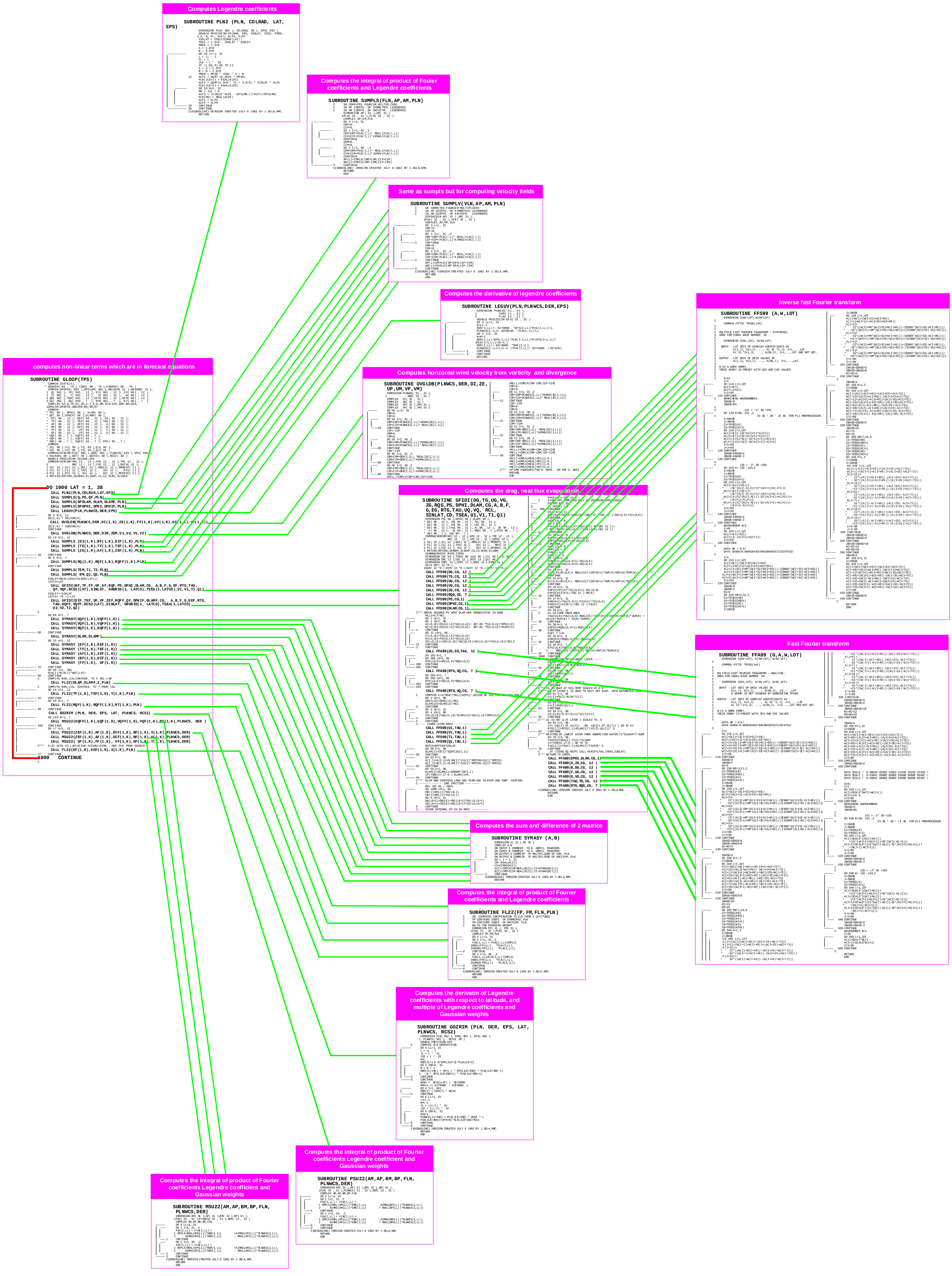
```
for i = 1 to n
   t = (A[i] + B[i]) / 2;
   C[i] = t * t;
```

  - Array

```
for i = 1 to n
   for j = 1 to n
      t[i] = (A[i] + B[i]) / 2;
   for j = 1 to n
      C[i, j] = t[i] * t[i];
```

- **Reduction**

```
for i = 1 to n
   sum = sum + A[i];
```



Computes Legendre coefficients
SUBROUTINE PLN2 (PLN, COLRAD, LAT, EPS)
Computes the integral of product of Fouier coefficients and Legendre coefficients
SUBROUTINE SUMPLS(FLN,AP,AM,PLN)
Same as sumpls but for computing velocity fields
SUBROUTINE SUMPLV(VLN,AP,AM,PLN)
Computes the derivative of legendre coefficients
SUBROUTINE LEGUV(PLN,PLNWCS,DER,EPS)
computes non-linear terms which are in forecast equations
SUBROUTINE GLOOP(TPS)
Computes horizontal wind velocity from vorticity and divergence
SUBROUTINE UVGLOB(PLNWCS,DER,DI,ZE, UP,UM,VP,VM)
Computes the drag, heat flux evaporation
SUBROUTINE GFIDI(DG,TG,UG,VG, ZG,RQG,PS,DPHI,DLAM,CG,A,B,F, G,EG,RTG,TAU,UQ,VQ, RCL, SINLAT,CD,TSEA,U1,V1,T1,Q1)
Inverse fast Fourier transform
SUBROUTINE FFS99 (A,W,LOT)
Fast Fourier transform
SUBROUTINE FFA99 (G,A,W,LOT)
Computes the sum and difference of 2 matrice
SUBROUTINE SYMASY (A,B)
Computes the integral of product of Fourier coefficients and Legendre coefficients
SUBROUTINE FL22(FP,FM,FLN,PLN)
Computes the derivatin of Legendre coefficients with respect to latitude, and multiple of Legendre coefficients and Gaussian weights
SUBROUTINE GOZRIM (PLN, DER, EPS, LAT, PLNWCS, RCS2)
Computes the integral of product of Fourier coefficients Legendre coefficient and Gaussian weights
SUBROUTINE PSU22(AM,AP,BM,BP,FLN, PLNWCS,DER)
Computes the integral of product of Fourier coefficients Legendre coefficient and Gaussian weights
SUBROUTINE MSU22(AM,AP,BM,BP,FLN, PLNWCS,DER)

Interprocedural Parallelization

- **Why? Amdahl's Law**
- **Interprocedural symbolic analysis**
  - Find interprocedural array indexes which are affine expressions of outer loop indices
- **Interprocedural parallelization analysis**
  - Data dependence based on summaries of array regions accessed
    - If the regions do not intersect, there is no parallelism
  - Find privatizable scalar variables and arrays
  - Find scalar and array reductions

Conclusions

- **Basic parallelization**
  - Doall loop: loops with no loop-carried data dependences
  - Data dependence for affine loop indexes
    = integer linear programming

- **Coarse-grain parallelism because of Amdahl's Law**
  - Interprocedural analysis is useful for affine indices
  - Asks users for help on unresolve dependence