The Data-Dependence graph of Adjoint Codes

Laurent Hascoët

INRIA Sophia-Antipolis

November 19th, 2012

Data-Dependence graph of this talk

Relate the parallel properties of a code to those of its adjoint

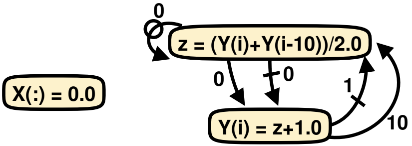
- **Data-Dependence Graphs (DDG's)**
  - Definition, Example
  - Construction, Usage
- **Adjoint codes**
  - Definition, Usage
  - Example
- **Refined DDG's for increments**
- **Adjoint DDG**
  - nodes
  - dependencies
- **Applications**

Data-Dependence Graphs (DDG's)

Given the source of a program,

- The DDG nodes are the operations of the program. Granularity can be chosen: ops, instructions, blocks ...
- Iterated DDG nodes are collapsed $\Rightarrow$ notion of distance in iteration spaces
- The DDG arrows are the necessary execution partial order. Collapsed nodes $\Rightarrow$ possible cycles!

```
X(:) = 0.0
DO  i=10,900
   z = (Y(i)+Y(i-10))/2.0
   Y(i) = z+1.0
ENDDO
```



How DDG's are built

Data-Dependencies originate from variables (*"data"*).

A variable `v` causes a Data-Dependency from node `N1` to node `N2` iff

- `N1` writes `v` and `N2` reads `v` (*"true dependency"*), or
- `N1` reads `v` and `N2` overwrites `v` (*"anti dependency"*), or
- `N1` writes `v` and `N2` overwrites `v` (*"output dependency"*).

and in addition (*but not always necessary...*) `v` is not totally overwritten between `N1` and `N2`.

What DDG's are for

A Data-Dependency from `N1` to `N2` means that any rescheduling of operations must keep the order `N1` before `N2`.
$\Rightarrow$ DDG's are central for parallelization.

- DDG's capture notions of dead code, inlining ...
- In loop parallelization, DDG cycles (SCC) cannot be parallelized nor vectorized.
- Loop nests can be transformed according to Data-Dependency distances.
- In Message-Passing, DDG cycles capture deadlocks.
- Scalar expansion is a classical way to lift anti and output dependencies, thus breaking cycles.

Automatic Differentiation's Adjoint codes

- Adjoints are the most efficient way to obtain analytical gradients.
- Uses abound: optimization, least-squares, parameter estimation, sensitivities, uncertainty quantification.
- Adjoints can be built mechanically through AD.

Adjoints propagate backwards the gradient of the result. Consider P

$$\mathtt{P} : \{I_1; I_2; \ldots I_{p-1}; I_p\}$$

that computes a function $F : X \mapsto Y$

$$F(X) = Y = V_p = f_p(f_{p-1}(\ldots f_2(f_1(V_0 = X)) \ldots))$$

its gradient is

$$\frac{\partial Y}{\partial X} = f'_p(V_{p-1}) \times f'_{p-1}(V_{p-2}) \times \cdots \times f'_2(V_1) \times f'_1(V_0)$$

that the adjoint code evaluates from left (row vector) to right.

Example

$$r = x*x + y*y$$

$$r = \mathrm{sqrt}(r)$$

$$x = x/r$$

$$y = y/r$$

$$\begin{aligned} \bar{r} &= \bar{r} - \bar{y}*y/(r*r) \\ \bar{y} &= \bar{y}/r \end{aligned}$$

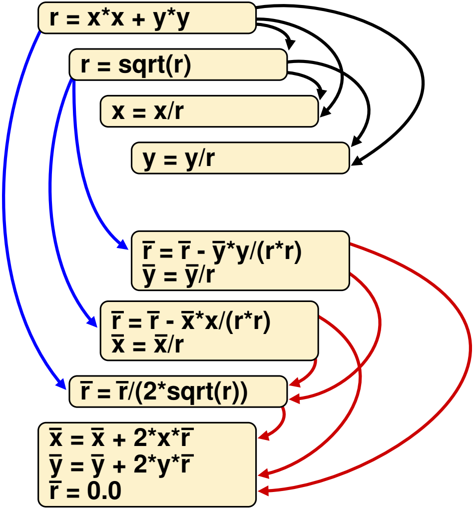
$$\begin{aligned} \bar{r} &= \bar{r} - \bar{x}*x/(r*r) \\ \bar{x} &= \bar{x}/r \end{aligned}$$

$$\bar{r} = \bar{r}/(2*\mathrm{sqrt}(r))$$

$$\begin{aligned} \bar{x} &= \bar{x} + 2*x*\bar{r} \\ \bar{y} &= \bar{y} + 2*y*\bar{r} \\ \bar{r} &= 0.0 \end{aligned}$$

- Two successive parts (sweeps)
- 2^{nd} sweep computes the gradient, reverse order.
- 1^{st} sweep computes needed original values $\Rightarrow$ copy of original code.
- Mechanism (not shown here) recovers needed `r`, `x`, `y`
- Gradient obtained in final $\bar{\mathtt{x}}$ and $\bar{\mathtt{y}}$.

What about the DDG ?



- Chosen granularity: I_k and $\overline{I_k}$
- Black deps are the original deps on the original copy.
- Blue deps show uses of direct values for the derivatives.
- Red deps caused only by derivative variables.
- Red deps seem symmetric of Black deps !?
  $\rightarrow$ true ? why ?

Two remarks

1: For DDG, increments behave just like reads

2: The adjoint of a read is an increment, and conversely.

1: Increments behave like reads

- Define the effect of DDG node on variable `v` to be ⓘ iff `v` is only used like `v = v +...`.
- Warning: make sure increments are atomic !

Successive increments are data-independent
$\Rightarrow$ refined cases for Data-Dependency:

	ⓝ	ⓡ	ⓘ	ⓦ
ⓝ				
ⓡ			•	•
ⓘ		•		•
ⓦ		•	•	•

2: Adjoint of read is increment, and vice-versa

- Suppose I_k (only) reads v (r)
- then $f'_k = \begin{pmatrix} \ddots & \bullet & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \ddots \end{pmatrix}$
- so that $\overline{I_k}$ actually executes:

$$\begin{pmatrix} \dots & \overline{\mathrm{v}} & \dots \end{pmatrix} = \begin{pmatrix} \dots & \overline{\mathrm{v}} & \dots \end{pmatrix} \times \begin{pmatrix} \ddots & \bullet & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \ddots \end{pmatrix}$$

- which implies $\overline{I_k}$ only increments $\overline{\mathrm{v}}$ (i)
  ... or just doesn't mention it (n)

Table of adjoint DDG nodes

- Looking at all cases:
  - $\mathtt{N}$ (n) on $\mathtt{v}$ $\Rightarrow$ $\overline{\mathtt{N}}$ {(n)} on $\overline{\mathtt{v}}$
  - $\mathtt{N}$ (r) on $\mathtt{v}$ $\Rightarrow$ $\overline{\mathtt{N}}$ {(i),(n)} on $\overline{\mathtt{v}}$
  - $\mathtt{N}$ (i) on $\mathtt{v}$ $\Rightarrow$ $\overline{\mathtt{N}}$ {(r),(n)} on $\overline{\mathtt{v}}$
  - $\mathtt{N}$ (w) on $\mathtt{v}$ $\Rightarrow$ $\overline{\mathtt{N}}$ {(w),(r),(i),(n)} on $\overline{\mathtt{v}}$
- which implies conversely:
  - $\overline{\mathtt{N}}$ (n) on $\overline{\mathtt{v}}$ $\Rightarrow$ $\mathtt{N}$ {(w),(r),(i),(n)} on $\mathtt{v}$
  - $\overline{\mathtt{N}}$ (r) on $\overline{\mathtt{v}}$ $\Rightarrow$ $\mathtt{N}$ {(w),(i)} on $\mathtt{v}$
  - $\overline{\mathtt{N}}$ (i) on $\overline{\mathtt{v}}$ $\Rightarrow$ $\mathtt{N}$ {(w),(r)} on $\mathtt{v}$
  - $\overline{\mathtt{N}}$ (w) on $\overline{\mathtt{v}}$ $\Rightarrow$ $\mathtt{N}$ {(w)} on $\mathtt{v}$

… and finally

$$\textbf{If} \quad \overline{I_k} \longrightarrow \overline{I_j}$$
$$\textbf{then} \quad I_j \longrightarrow I_k$$

- In other words, the adjoint DDG is equal to (or smaller than) the reversed original DDG
- Distances are preserved
- Nature of dependences (*true*, *anti*, *output*) is not preserved

Conclusion, Applications

- The adjoint DDG has the same structure as the original DDG
- Most parallel properties can propagate to the adjoint code:
  - The adjoint of a vector code is a vector code (in most cases).
  - The adjoint of a loop with independent iterations has independent iterations.
- There can even be more parallelism in the adjoint !
- ... but keep in mind the hypotheses (atomic increments).

Each HPC paradigm (e.g. Message-Passing SPMD) can use this DDG property in a specific way