High Order Automatic Differentiation with MPI

Mu Wang and Alex Pothen

Computer Science Department, Purdue University, West Lafayette, U.S

PURDUE UNIVERSITY®

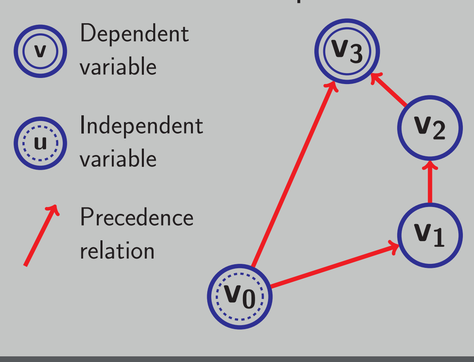
Introduction

- Automatic Differentiation (AD) is an algorithmic technique for augmenting computer programs to compute derivatives accurately and efficiently.
  - ▷ Analytic technique, so no truncation errors, and machine precision accuracy is obtained.
  - ▷ Relies on the fact that the computation of a mathematical function is a sequence of elementary/intrinsic operations.

$$\mathbf{v_i} = \varphi_i(\mathbf{v_j})_{\{\mathbf{v_j}:\mathbf{v_j}\prec\mathbf{v_i}\}}, \mathbf{i} = 1, 2, 3, \cdots, \mathbf{l}$$

- Primitives in AD
  - ▷ Declare independent variables: the initial values are pre-determined.
  - ▷ Declare dependent variables: the derivatives need to be computed.

Code	SAC Sequence
Independents: $\mathbf{x}$	`Indep:` $\mathbf{v_0}$
$\mathbf{a = x * x}$	$\mathbf{v_1 = v_0 * v_0}$
$\mathbf{t = a * a}$	$\mathbf{v_2 = v_1 * v_1}$
$\mathbf{y = x * y}$	$\mathbf{v_3 = v_0 * v_2}$
Dependent: $\mathbf{y}$	`Dep:` $\mathbf{v_3}$



Reverse Mode of AD

- A *Reverse* mode AD algorithm evaluates the derivatives in the *opposite* order from the evaluation of the SAC sequence.
  - ▷ In each step, the algorithm computes the derivatives of a current equivalent function.
  - ▷ The equivalent function is defined by the processed SACs : $\varphi_1, \cdots, \varphi_i$.
  - ▷ The complexity is proportional to the number of dependent variables.
  - ▷ The SAC sequence has to be stored on a *computation trace*.
- First order reverse mode AD is also called adjoint mode.
  - ▷ In each step, the adjoint of a variable is the partial derivative of the current equivalent function w.r.t that variable.
- It's promising to apply Reverse mode AD for an MPI program because:
  - ▷ Each processor can store the locally evaluated SACs in its own memory.
  - ▷ Reverse mode can exploit the sparsity of the derivatives automatically.
- More information available at :

Related Work

- Many papers have been written on developing first order reverse mode for an MPI program.
  - ▷ Paul Hovland : "AD of parallel programs", 1997.
    - – Describes key issues that must be addressed for parallel programs.
  - ▷ Christèle Faure and Patrick Dutto : "Extension of Odyssee to the MPI library – Reverse mode",1999.
    - – First order reverse mode for `Fortran 77`.
  - ▷ Jean Utke, et al : "Toward adjoinable MPI", 2009.
    - – A tool-indedendent, adjoinable MPI wrapper library.
- The key ideas of previous work:
  - ▷ Consider a `Send/Recv` pair as a remote value assignment.
  - ▷ Reverse the communication: a `Send/Recv` pair for values becomes a `Recv/Send` pair for adjoints (derivatives).
- Unfortunately, this approach can not be applied to second or higher orders.
  - ▷ The intermediate value(s) associated with a variable is no longer an adjoint value, but a slice of the current derivative tensor.

Our Contribution

- We describe a new parallel Reverse mode algorithm to compute higher order derivative using an MPI program.
  - ▷ The innovation: a `Send/Recv` pair should not be considered only as a remote value assignment, but as dummy independent/dependent variables.
- The innovation permits communication to be treated analogous to computation. Hence communication does not need to be reversed, but can be re-performed in a separate phase.
  - ▷ Avoids the complications created by reversing the communications.

Send/Recv Primitives in Reverse Mode

- In AD semantics, a `Send` or a `Recv` is similar to declaring a dependent or an independent variable.
  - ▷ A `Send` creates a *dummy dependent* variable: the derivative of the variable is needed (by another process).
  - ▷ A `Recv` creates a *dummy independent* variable: the value of the variable is "pre-determined" (by another process).

The New MPI Reverse Mode Algorithm

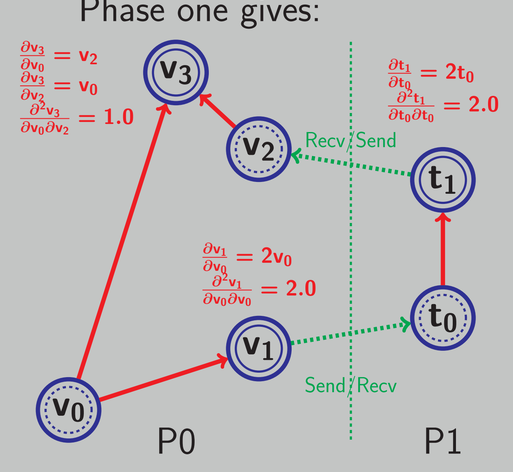
- In addition to the computation trace, the MPI Reverse mode needs a *communication trace*
  - ▷ Contains the information of all `Send/Recv` pairs.
  - ▷ For collective communications, break them down to basic `Send/Recv` communications (as in earlier work).
- The evaluation of the derivatives has two phases:
  - ▷ Phase one: run the high order reverse mode on each process from the local computation trace.
    - ▸ Evaluate the derivative tensor for each dependent variable (including the dummy ones).
  - ▷ Phase two: redo the communication in *forward* order from the communication trace.
    - ▸ On `Send`: send the derivatives of the dummy dependent variable.
    - ▸ On `Recv`: evaluate the derivatives for every dependent variable by treating the *dummy independent variable* as a function defined by the received derivative information.

Example : A Simple MPI Program on Two Processors

- For objective function defined by:

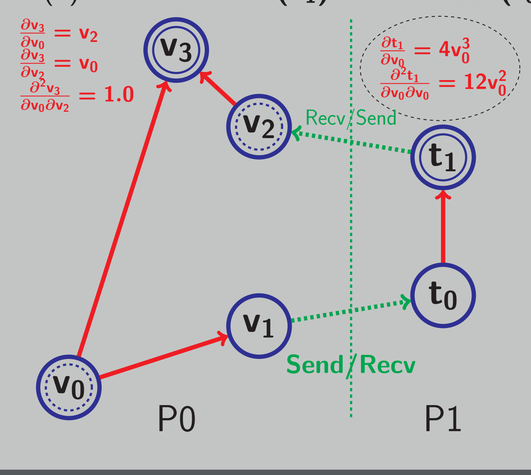
P0	P1
`Indep:` $\mathbf{v_0}$	
$\mathbf{v_1 = v_0 * v_0}$	
$\mathbf{Send(v_1)}$	$\mathbf{Recv(t_0)}$
	$\mathbf{t_1 = t_0 * t_0}$
$\mathbf{Recv(v_2)}$	$\mathbf{Send(t_1)}$
$\mathbf{v_3 = v_0 * v_2}$	
`Dep:` $\mathbf{v_3}$	

Phase one gives:

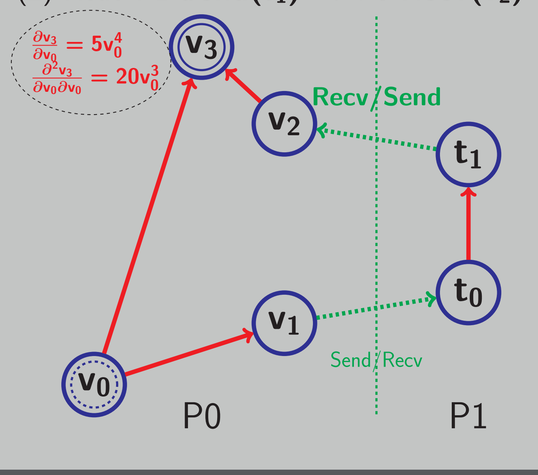


- Phase two gives:

(1) Redo P0 $\mathbf{Send(v_1)} \rightarrow$ P1 $\mathbf{Recv(t_0)}$



(2) Redo P1 $\mathbf{Send(t_1)} \rightarrow$ P0 $\mathbf{Recv(v_2)}$



Case Study: Parallel Hessian for Separable Functions:

Test platform: Intel i5-2400 quad-core processor with 4GB memory on each node.

- Rosenbrock function: $\mathbf{f(x)} = \sum_{i=1}^{N-1} \left[\mathbf{a(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2}\right]$

N=512,000 :

P =	1	2	4	8	16	32	64
Function Eval Time	0.47	0.30	0.20	0.16	0.15	0.13	0.15
Hessian EvalTime	8.73	3.31	2.24	1.36	1.04	0.79	0.67
Speed up	1	2.63	3.90	6.42	8.39	11.05	13.02

- Thomson problem: $\mathbf{f(\theta, \phi)} = \sum_{i<j} \frac{1}{|(\theta_i,\phi_i)-(\theta_j,\phi_j)|}$, (θ, ϕ) is spherical coordinate on unit sphere. $|(\theta_i, \phi_i) - (\theta_j, \phi_j)|$ is the Euclidean distance.

N=1,600 :

P =	1	2	4	8	16	32	64
Function Eval Time	3.40	1.69	0.87	0.48	0.27	0.18	0.16
Hessian Eval Time	108.9	60.50	28.33	14.04	7.36	4.11	2.59
Speed up	1	1.80	3.84	7.76	14.80	26.50	42.05