# Productive Parallel Programming for the Masses

Christian Grothoff
christian@grothoff.org

University of Denver

# Parallel Computing is Mainstream

- **Desktop/Pentium Xeon:** hyperthreading, SMP
- **Notebook/Core Duo:** 2 cores
- **Playstation 3/Cell:** 9 processing units
- **Supercomputer/Blue Gene:** 128k processors

Programming these systems well is hard, even at 50% of peak!

# The Problem: Developing Parallel Stream Applications

- Developers know how to write sequential code
- Parallel programing is error-prone
- High-performance parallel programming is really hard
- With GPUs for $4,000, we could have 2,600 cores...

  $\Rightarrow$ Developers much more expensive than hardware

# X10 vs. the DUP System[1]

## X10

10x faster, 10x as productive in 10 years for BlueGene

## DUP

$\frac{1}{2}$ the speed, 10x as productive in 10 months for POSIX

[1] Available at `http://dupsystem.org/`

# A Blast from the Past: CMS Pipelines

- Similar to UNIX pipes
- Sligthly different syntax
- NEW: multistream pipelines

See also: CMS Pipelines User's Guide [7]

# Example: CMS Pipeline

```
Pipe < INPUT FILE A % input is a stage!
|   drop 4           % like ‘‘eat 4’’
| sort 34-36         % sort by colums 34-36

| > OUTPUT FILE B    % output is a stage!
```

# Example: CMS Multistream Pipeline

```
Pipe < INPUT FILE A
| d:drop 4   % label stage ‘‘d’’
| sort 34-36 %
| i:faninany % label stage  ‘‘i’’
| > OUTPUT FILE B
?            % end of primary pipeline
d:           % take 2nd output of ‘‘d’’
| i:         % make it the 2nd input of ‘‘i’’
```

# Limitations of CMS Pipeines

- Sequential execution on one CPU, no parallelism
- Only available on CMS and z/OS
- Record-oriented (CMS is a mainframe OS)

... but these are easy to address!

# Our Solution: DUP ≡ Distributed Multi-Stream Pipelines

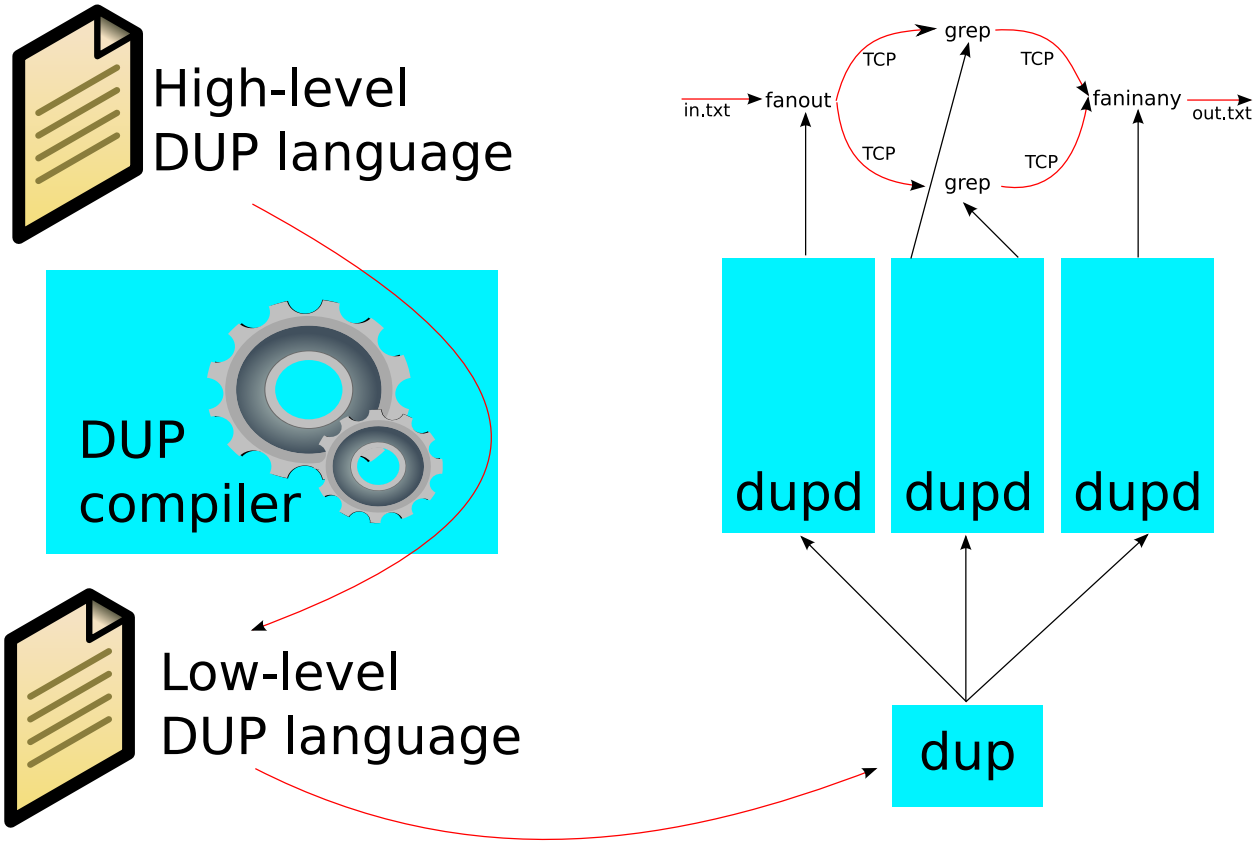
- Computation composed of stages in a flow-graph
- Stages run as individual processes in parallel
- Stages can have any number of inputs and outputs
- DUP used to connect stages
- DUP provides stages for common problems

⇒ Simple stream-oriented parallel programming model that also guides developers towards modular design

# Example: DUP “Assembly”

```
dup <<EOF
drop@localhost[0<file.a,1|sort:0,3|merg:3] $ drop 4  ;
sort@localhost[1|merge:0]                  $ sort    ;
merg@localhost[1>file.b]                   $ faninany;
EOF
```

# DUP Architecture



# Vision: DUP High-level Language

```
import duplib;

$in = read("file.a");
($body, $head) = drop ($in, "4");
write (faninany (sort ($body),
                 $head),
        "file.b");
```

# DUP Limitations

- Stages communicate via streams

  $\Rightarrow$ Computation must be stream-oriented

- Stages run in parallel, internals are up to the stage

  $\Rightarrow$ DUP parallelism limited by stages

# DUP Application Domains

- Genome sequence processing
- Discrete event simulation
- Intrusion Detection
- Video conferencing
- Event surveillance
- System administration
- ...

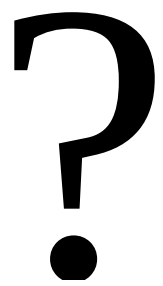
# Related Work

- InfoSphere Streams [1] & Dryad [8]
- StreamFlex [10] & StreamIt [11]
- Kahn Process Networks [9]
- Linda [5]

# Future Work

- High-level DUP programming language (will be an aspect-oriented coordination mini-language)
- Develop more filters/stages and applications
- Type systems for streams (see also: SPADE [6])
- Add common features of distributed systems [2, 3] while maintaining **simplicity**, **portability** and **language independence**

# Questions



# Copyright

# CMS Pipeline Terminology

- Stage – Program that accomplishes a specific task
- Stage Separator – |
- Stream – flow of data into and out of a stage
- Device Driver – stage that interfaces with the environment
- Filter – processes data without interfacing with environment

# Common Filters in CMS

**locate, find, nlocate, nfind:** select records with specified target

**between, inside, outside, ninside:** select records between specified targets

**take, drop:** select records by counter

**combine, overlay:** combine records

$\Rightarrow$ roughly equivalent to UNIX filters

# Pipeline Stalls

Multistream pipelines introduce a new potential problem:

- Every stage might be waiting for some other stage to perform some function (read or write)
- Cause is usually stage that reads multiple inputs *in a particular order* (or multiple records)
- Preceding stages may not be able to deliver order or quantity required

When a stall occurs, you receive a return code of "-4095".

# CMS Multistream Pipelines

- Multistream pipelines are pipelines that contains stages that have multiple input or output streams
- Implement primary pipeline; place a label on every stage with multiple input or output streams
- Use the endchar "?" to indicate the end of the primary pipeline
- Write the next pipeline, using the labels to refer to streams from the primary pipeline

# References


[1] Lisa Amini, Henrique Andrade, Ranjita Bhagwan, Frank Eskesen, Richard King, Philippe Selo, Yoonho Park, and Chitra Venkatramani. Spc: A distributed, scalable platform for data mining. In *Workshop on Data Mining Standards, Services and Platforms (DM-SPP)*, 2006.

[2] Hari Balakrishnan, Magdalena Balazinska, Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Eddie Galvez, Jon Salz, Michael Stonebraker, Nesime Tatbul, Richard Tibbetts, and Stan Zdonik. Retrospective on aurora. *The VLDB Journal*, 13(4):370–383, 2004.

[3] Magdalena Balazinska, Hari Balakrishnan, Samuel R. Madden, and Michael Stonebraker. Fault-tolerance in the borealis distributed stream processing system. *ACM Trans. Database Syst.*, 33(1):1–44, 2008.

[4] Ian Buck. Gpu computing: Programming a massively parallel processor. In *Proceedings of the International Symposium on Code Generation and Optimization*, 2007.

[5] Nicholas Carriero and David Gelernter. Linda in context. *Commun. ACM*, 32(4):444–458, 1989.

[6] Martin Hirzel, Henrique Andrade, Bugra Gedik, Vibhore Kumar, Giuliano Losa, Robert Soule, and Kun-Lung-Wu. Spade language specification. Technical report, IBM Research, March 2009.

[7] IBM. *CMS Pipelines User's Guide*. IBM Corp., http://publibz.boulder.ibm.com/epubs/pdf/hcsh1b10.pdf, version 5 release 2 edition, Dec 2005.

[8] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *European Conference on Computer Systems (EuroSys)*, pages 59–72, Lisabon, Portugal, March 2007.

[9] Gilles Kahn. The semantics of a simple language for parallel programming. *Information Processing*, (74):993–998, 1974.

[10] Jesper H. Spring, Jean Privat, Rachid Guerraoui, and Jan Vitek. Streamflex: high-throughput stream programming in java. *SIGPLAN Not.*, 42(10):211–228, 2007.

[11] William Thies, Michal Karczmarek, and Saman P. Amarasinghe. Streamit: A language for streaming applications. In *CC '02: Proceedings of the 11th International Conference on Compiler Construction*, pages 179–196, London, UK, 2002. Springer-Verlag.