Spass mit paralleler und verteilter Programmierung

Christian Grothoff
christian@grothoff.org

Technische Universität München

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
- We will soon have 100s of cores on mainstream CPUs

  $\Rightarrow$ Developers much more expensive than hardware

A Blast from the Past: CMS Pipelines

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

See also: CMS Pipelines User's Guide [5]

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!
```

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

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’’
```

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".

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:

`http://dupsystem.org/`

Our Solution: DUP ≡ Distributed Multi-Stream Pipelines

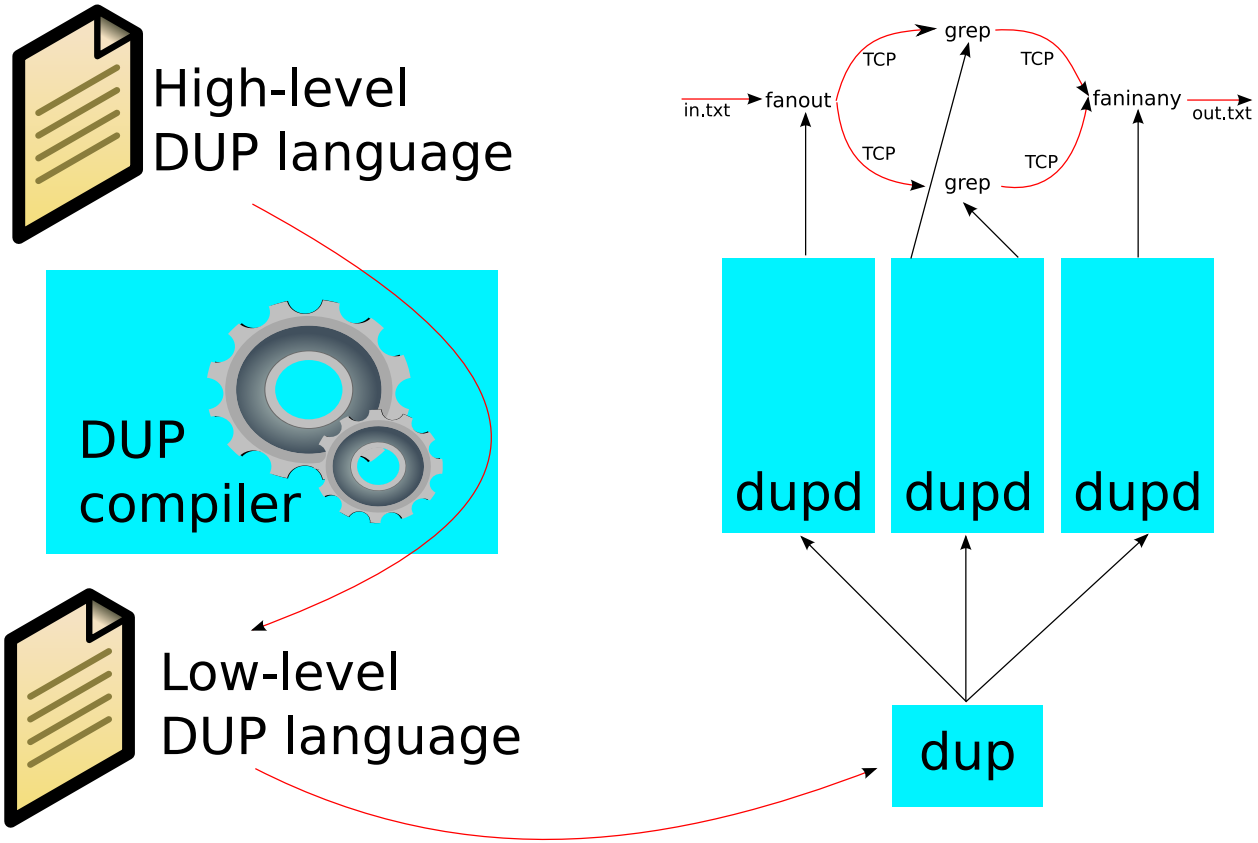
- Computation composed of stages in a flow-graph
- Stages run as individual processes in parallel
- Stages can have any number of inputs and outputs
- DUP used to connect stages
- DUP provides stages for common problems

⇒ 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
- ...

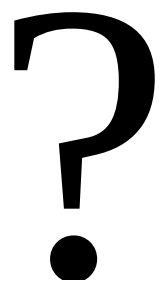
Fun with DUP

```
nodegen@localhost:30001[1|split:0]$ cat RowRowRowYourBoat.mp3;
split@localhost:30001[5|play1:0,1|d1:0,3|d2:0,6|d3:0,7|d4:0,8|d5:0]$ fanout 50;
d1@localhost:30001[1|play2:0]$ delay 1000;
d2@m1:30001[1|play3:0]$ delay 2000;
d3@m2:30001[1|play4:0]$ delay 3000;
d4@m3:30001[1|play5:0]$ delay 4000;
d5@m4:30001[1|play6:0]$ delay 5000;
play1@localhost:30001[4>/dev/null]$ mpg123 -;
play2@m1:30001[4>/dev/null]$ mpg123 -;
play3@m2:30001[4>/dev/null]$ mpg123 -;
play4@m3:30001[4>/dev/null]$ mpg123 -;
play5@m4:30001[4>/dev/null]$ mpg123 -;
play6@m5:30001[4>/dev/null]$ mpg123 -;
```

Future Work

- Remove "arbitrary code execution" feature
- High-level DUP programming language
- Develop more filters/stages and applications
- Type systems for streams (see also: SPADE [4])
- Add common features of distributed systems [1, 2] while maintaining **simplicity**, **portability** and **language independence**

Questions



Copyright

References

[1] 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.

[2] 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.

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

[4] 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.

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