Distributed Stream Processing with the DUP System

Christian Grothoff
christian@grothoff.org

Parallel Computing is Mainstream

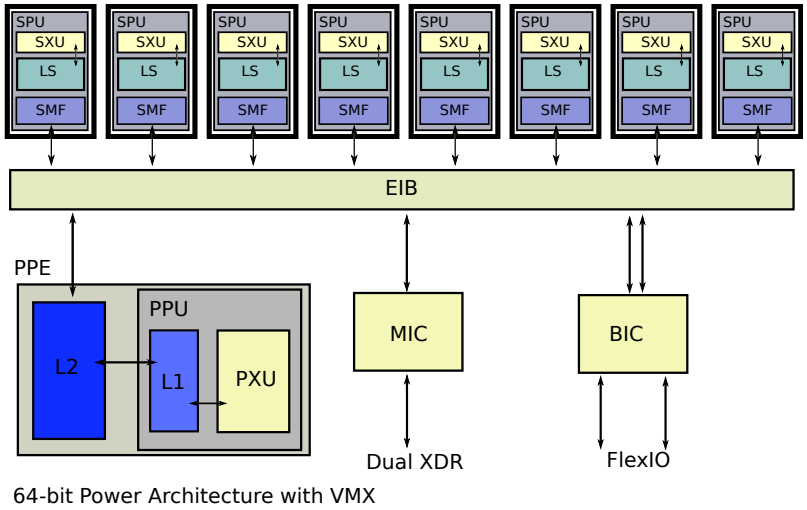
- **Desktop/Pentium Xeon:** hyperthreading, SMP
- **Notebook/Core Duo:** 2 cores
- **Playstation 3/Cell:** 9 processing units
- **Supercomputer/Blue Gene:** 128k processors

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

The Problem

- Most developers (only) know how to write sequential code
- Parallel programming is error-prone
- High-performance parallel programming is really hard
- Large amounts of legacy code are not parallel

$\Rightarrow$ Developers more expensive than hardware



Cell Processor

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

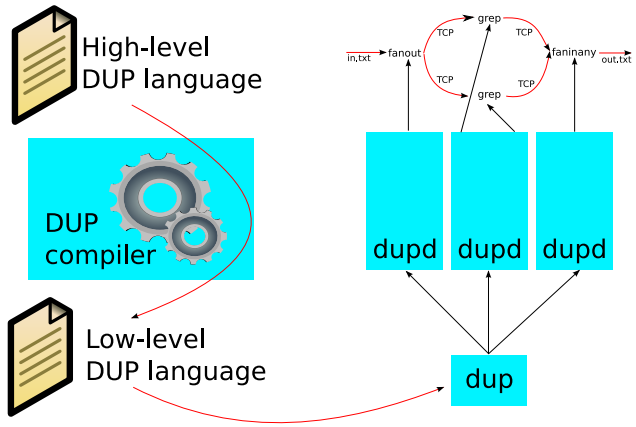
- Sequential execution on one CPU, no parallelism
- Only available on CMS and z/OS
- Record-oriented (CMS is a mainframe OS)

... but these are easy to address!

Our Solution: Distributed Multi-Stream Pipelines

- Computation composed of stages in a flow-graph
- All stages run as individual processes in parallel
- DUP Runtime connects stages
- DUP Runtime Library provides stages for common problems

$\Rightarrow$ Eliminates common problems with parallel programming and guides developers towards modular design



DUP Architecture

```
dup <<EOF
d@localhost[1|s:0,3|m:3] $ drop 4   ;
s@localhost[1|m:0]       $ sort     ;
m@localhost[1>output]    $ faninany;
EOF
```

DUP Assembly

Vision: High-level DUP Language

```
import duplib;

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

Vision: High-Level DUP Language

Aspects	**aspect** encrypt-network-streams ($\$key$, $\$iv$): **around stream** (_@$\$srchost$, _@$\$dsthost$) **where** $\$srchost \neq \$dsthost$ **before** openssl enc -e -aes -K $\$key$ -iv $\$iv$ **after** openssl enc -d -aes -K $\$key$ -iv $\$iv$
Types	**type** openssl enc -e -aes -K $\$key$ -iv $\$iv$: **in** 0: $\langle T \rangle$ **out** 1: 'AES $\langle T \rangle$ [$\$key$, $\$iv$] **type** openssl enc -d -aes -K $\$key$ -iv $\$iv$: **in** 0: 'AES $\langle T \rangle$ [$\$key$, $\$iv$] **out** 1: $\langle T \rangle$

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

State of the Art and DUP

Approach (Representative)	Speed	Prod.	Migr.	Domain
API (MPI)	+++	o	o	+
Extend (UPC)	+	-	o/- - -	+
Scratch (X10, Fortress)	?	+	- - -	++
Domain-Extend (Spade)	++	+	+/- - -	+
Domain-Embrace (DUP)	+	+++	+++	+++

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**

DUP Application Domains

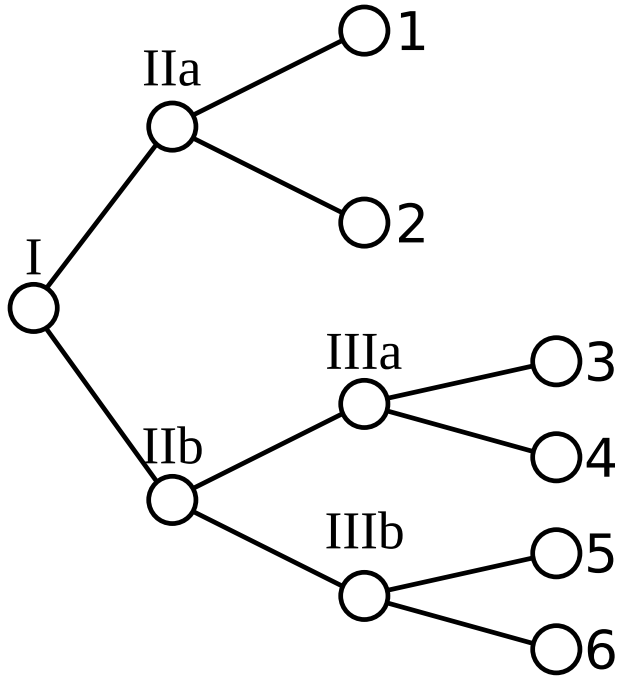
- Genome sequence processing
- Discrete event simulation
- Intrusion Detection
- Video conferencing
- Event surveillance
- System administration
- ...

Case Study: ARB

Biological Questions

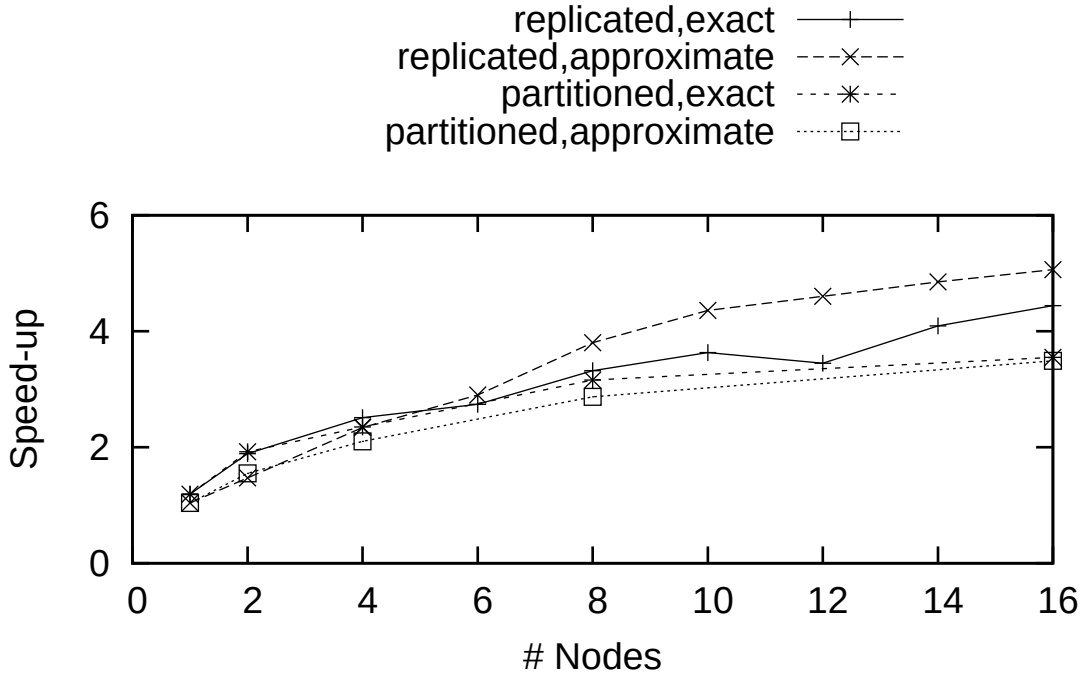
- Which are the most specific OSSs for a species?
- Which are the most specific OSSs for a subtree in the phylogenetic tree?

Input: Phylogenetic Tree

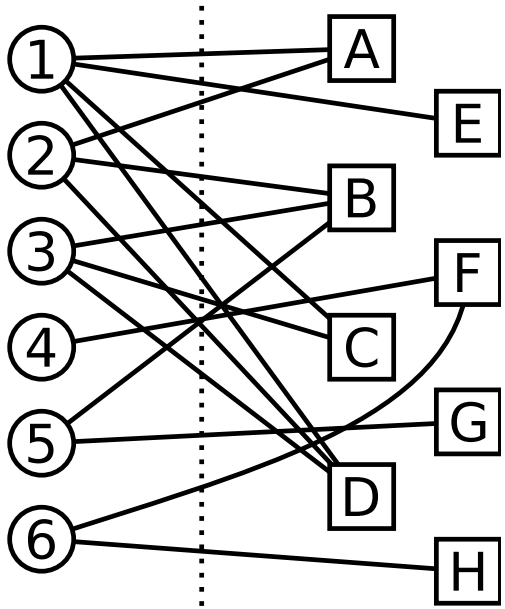


Parallel Mapping of Primers to Species

```
s @opt1:88[0<in.txt,1|p1:0,3|p2:0] $ fanout;
p1@opt1:88[1|pe:0]    $ arb_probe_dup;
p2@opt2:88[1|pe:3]    $ arb_probe_dup;
pe@opt2:88[1>out.txt] $ gather;
```



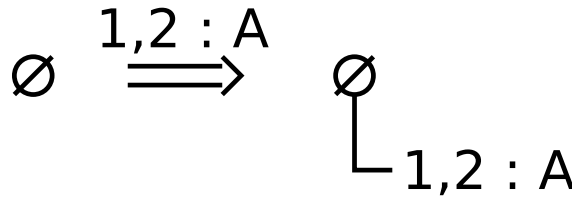
Intermediate Result: Species and OSS



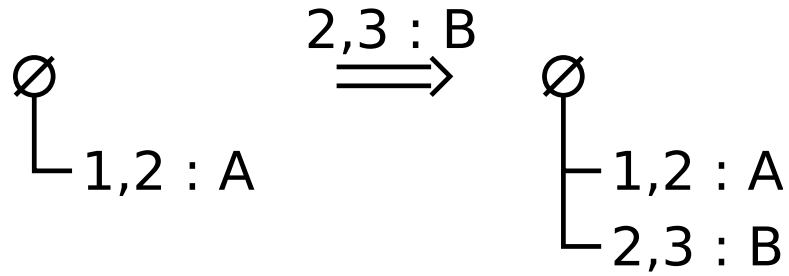
Biological Questions

- Which are the most specific OSSs for a species?
- Which are the most specific OSSs for a subtree in the phylogenetic tree?
- Sequence information may contain errors; allow up to k out-group hits!
- Perfect OSS may not exist even with out-group hits; maximize number of in-group hits

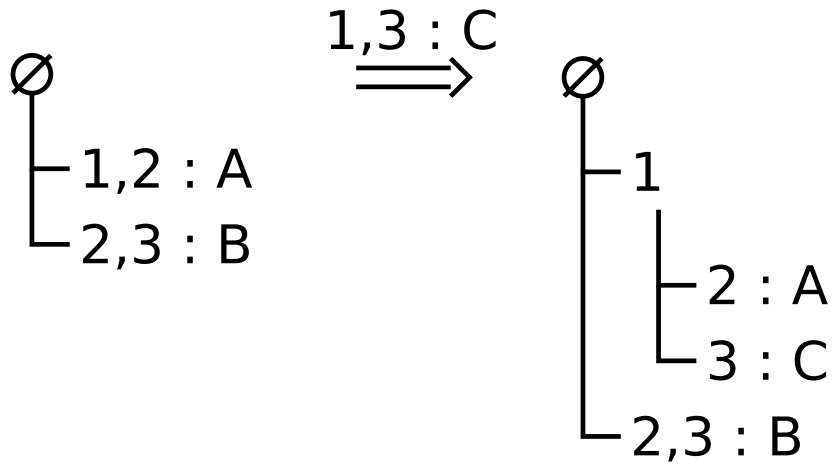
BGRT Creation (1/5)



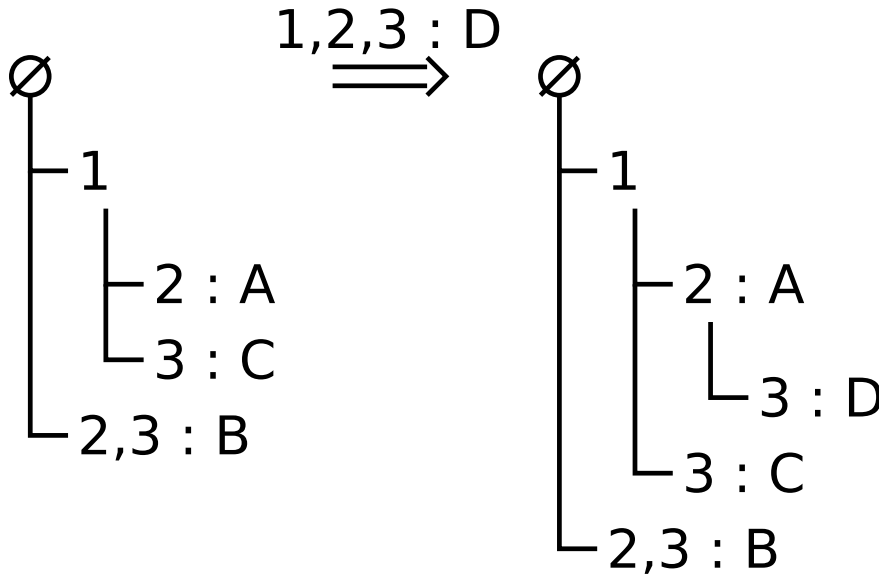
BGRT Creation (2/5)



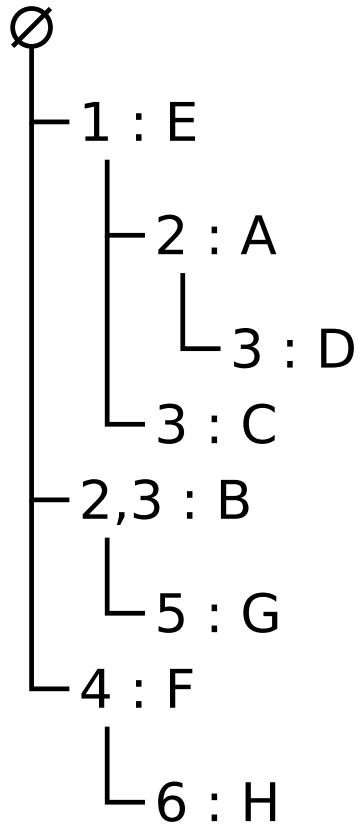
BGRT Creation (3/5)



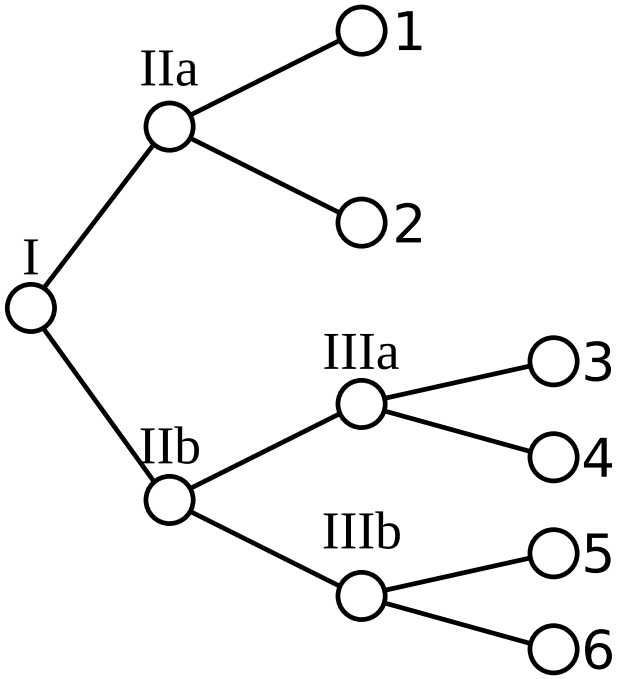
BGRT Creation (4/5)

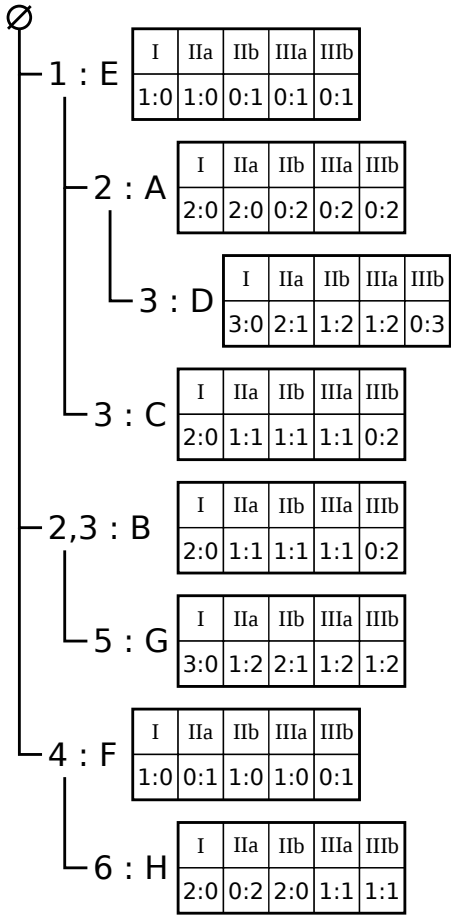


BGRT Creation (5/5)



Iterate-And-Bound





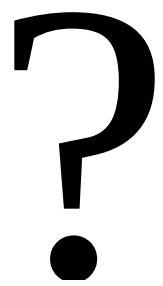
Final Result

Mismatches	Phase I	IIa	1	2	IIb	IIIa	3	4	IIIb	5	6
0	3:D,G	2:A	1:E	%	2:H	1:F	%	1:F	%	%	%
1	%	2:D^{+}	1:A,C^{+}	1:A,B	2:G^{+}	1:B,C,H^{+}	1:B,C	1:H^{+}	1:H	%	1:H
2	%	1:G^{*}	1:D^{+}	1:D,G^{+}	1:D^{*}	1:D,G^{+}	1:D^{+}	%	1:G^{+}	1:G	%
3	%	%	%	%	%	%	%	%	0:D^{*}	%	%

"$+$" Should probably not be computed (mismatches $>$, matches $=$)

"$*$" Even more useless (mismatches $>$, matches $<$)

Questions



Copyright

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.