lnec

PARALLEL PROGRAMMING IN GO FOR PERFORMANCE WITH THE PARGO LIBRARY

PASCAL COSTANZA

WHAT IS PARGO?

- Pargo is a library for parallel programming in Go at imec's ExaScience Lab:
 - based on our experiences with parallel programming in C++, Common Lisp, and Java
 - released under a BSD-style open source license at <u>https://github.com/exascience/pargo</u>
- Pargo supports numerous common parallel programming patterns:
 - Divide-and-conquer task-based parallelism
 - Parallel ranges, parallel reduction, parallel Boolean functions
 - Speculative parallelism
 - Parallel Quicksort and Mergesort
 - Parallel hash table
 - Parallel pipelines inspired by Java Parallel Streams introduced in JDK 8
 - including support for contexts, cancellation, and Go-style error handling

CONCURRENCY VS. PARALLELISM

- Concurrency is part of the problem domain.
- Needs solution even without multicore/node.
- Go is really good at this!

SELECT DESIRED SEAT BY CLICKING ON THE ABOVE CHART

- Parallelism is part of the solution domain.
- Only needed for performance.
- Pargo is really good at this! ;)



Gary W. Sabot, The Paralation Model, The MIT Press, 1988

ເກາຍc

https://xkcd.com/726/







ເຫາຍດ





ເງຍອ



ເຫາຍດ



ເຫາຍດ

DIVIDE-AND-CONQUER TASK-BASED PARALLELISM



ເງຍອ

DIVIDE-AND-CONQUER TASK-BASED PARALLELISM



DIVIDE-AND-CONQUER TASK-BASED PARALLELISM WITH 16 CORES



DIVIDE-AND-CONQUER TASK-BASED PARALLELISM WITH 4 CORES



DIVIDE-AND-CONQUER TASK-BASED PARALLELISM WITH LOAD IMBALANCE



DIVIDE-AND-CONQUER TASK-BASED PARALLELISM

- Task-based parallelism allows flexible distribution of work over CPU cores.
- Distributing work evenly over cores is often not optimal, because of load imbalance.

DIVIDE-AND-CONQUER TASK-BASED PARALLELISM

- Task-based parallelism allows flexible distribution of work over CPU cores.
- ...but how are tasks actually scheduled over the cores?



- Work stealing is known to be optimal both in theory and practice
 - Blumofe, Leiserson: Scheduling Multithreaded Computations by Work Stealing, Journal of the ACM, 1999
 - Frigo, Leiserson, Randall: The Implementation of the Cilk-5 Multithreaded Language, PLDI'98
- Successfully implemented in many languages and libraries:
 - Cilk for C;Threading Building Blocks for C++; Java fork/join; ...
 - ...and Go



Wonder Gopher by Ashley McNamara, https://github.com/ashleymcnamara/gophers

WORK STEALING FINDS OPTIMAL DISTRIBUTION ON THE FLY



ເງຍອ

8		fun	c sum(numbers []float64) float64 {
9			<pre>if len(numbers) < threshold {</pre>
10			var sum float64
11			<pre>for _, number := range numbers {</pre>
12			sum += number
13			}
14			return sum
15		2	}
16			half := len(numbers) / 2
17			var wg sync.WaitGroup
18			wg.Add(delta: 1)
19			var left float64
20			go func() {
21			<pre>defer wg.Done()</pre>
22	٢		<pre>left = sum(numbers[:half])</pre>
23			}()
24	٢		right := sum(numbers[half:])
25			wg.Wait()
26			return left + right
27]}	









...AND LOTS OF OTHER PARALLEL PROGRAMMING PATTERNS

- Parallel Do
- Parallel range
- Parallel reduction over int, float64, string, interface{}
- Parallel range reduction over int, float64, string, interface{}
- Parallel And, Or, RangeAnd, RangeOr
- Speculative variants of many of the above functions
- Sequential variants for debugging
- Parallel Quicksort and merge sort
- A parallel hash table (similar to Go's sync.Map)
- ...and parallel pipelines.

ເກາຍດ



່ເກາຍເ



ເຫາຍເ



ເຫາຍດ



ເຫາຍດ



ເງຍອ



ເງຍອ



ເງຍ



ເງຍ



ເງຍ

- Predefined pipeline sources for arrays, slices, strings, channels, and bufio.Scanner.
- Support for user-defined sources through the pipeline.Source interface.
- Support for several kinds of nodes (stages):
 - Sequential, ordered, parallel
 - Strictly ordered, limited parallel
 - Skip and Limit nodes
- Support for several kinds of filters:
 - Generic receive and finalize
 - Boolean filters: Every, Some, NotEvery, NotAny
 - Counting filter
 - Slice filter
- Support for contexts, including cancellation
- Support for error handling, including cancellation on error
- Support for fine-tuning of batch sizes

ເງຍອ

ELPREP: A HIGH-PERFORMANCE TOOL FOR SEQUENCING

- High-performance tool for preparing SAM files for variant calling.
- Multi-threaded application that runs entirely in RAM and merges multiple steps to avoid repeated file I/O.
- Can improve performance by a factor of up to x10 compared to standard tools.
- elPrep implemented in Go since version 3.0
 - https://github.com/exascience/elprep

	DNE	PU	BLISH	ABOUT	BROWSE	SEARCH	Q
							advanced searc
) OPEN ACCESS 🔌 PEER-REVIE RESEARCH ARTICLE	WED					41 Save	5 Citation
elPrep: High-Perf Files for Variant (ormance Prepara Calling	tion of Sequenc	e Align	ment/N	Лар	4,601 View	19 Share
Charlotte Herzeel 👓 🖾, Pasc	al Costanza 🚥, Dries Decap, J	an Fostier, Joke Reumers					
Published: July 16, 2015 • htt	ns://doi.org/10.1371/journal.pon	e 0132868					
Article Au	ithors Metrics	Comment	S	Media Co	overage	Downloa	d PDF 🔻
Abstract	Abotroot						
Introduction	Abstract					Cher	ck for updates
Implementation	elPrep is a high-performan	ce tool for preparing sequend	e alignmen	t/map files for	r variant calling	ADVER	TISEMENT
Methods	preparation steps such and	can be used as a replacemel	licatos roo	ools and Pical	ra tor e and eo on		
	while producing identi-						
Results	while producing identit	sort b	y coordina	ates 📕 fil	ter unmappe	d reads 📒 I	mark duplicate
Results Related Work	executing preparation	add re	y coordina ad group	ates 📕 fil s 📕 fil	ter unmappe ter sequence	d reads 📃 i dictionary 📕 i	mark duplicate merged
Results Related Work Conclusions and Future Work	while producing identity executing preparation many preparation ster application that runs e of several preparation preparation pineline of	e sort b add re Picard/Samtor	y coordina ead group	ates ∎ fil s ■ fil	ter unmapper ter sequence	d reads	mark duplicate merged
Results Related Work Conclusions and Future Work Supporting Information	while producing identity executing preparation many preparation step application that runs e of several preparation preparation pipeline o execution time from al	Picard/Samtod	y coordina ad group	ates ■ fil s ■ fil	ter unmapper ter sequence	d reads	mark duplicate:
Results Related Work Conclusions and Future Work Supporting Information Acknowledgments	while producing preparation many preparation step application that runs e of several preparation preparation pipeline of execution time from al about 15 minutes whe threads and 23GR of i	Picard/Samtor	y coordina ead group bls	ates ■ fil s ■ fil	ter unmapped ter sequence	d reads	mark duplicate: merged
Results Related Work Conclusions and Future Work Supporting Information Acknowledgments Author Contributions	when producing internation executing preparation many preparation step application that runs e of several preparation preparation pipeline of execution time from al about 15 minutes whe threads and 23GB of 1 reduces the runtime fr	eIPr	y coordina ead group bls	ates ∎ fil s ■ fil	ter unmapped ter sequence	d reads	mark duplicate merged
Results Related Work Conclusions and Future Work Supporting Information Acknowledgments Author Contributions References	when producting preparation many preparation step application that runs e of several preparation preparation pipeline of execution time from a about 15 minutes whe threads and 23GB of 1 reduces the runtime fr sequencing data for h computing time, and t	elPr	ep	ates ∎ fil	ter unmapped ter sequence	d reads dictionary	mark duplicate merged
Results Related Work Conclusions and Future Work Supporting Information Acknowledgments Author Contributions References Reader Comments (1) Media Coverage (0)	when producing preparation many preparation step application that runs e of several preparation preparation pipeline o execution time from at about 15 minutes whe threads and 23GB of reduces the runtime fr sequencing data for h computing time, and t Figures	elPrep (merge	ep dd)	ttes i fil	ter unmapper ter sequence	dictionary	mark duplicate merged
Results Related Work Conclusions and Future Work Supporting Information Acknowledgments Author Contributions References Reader Comments (1) Media Coverage (0) Figures	when producing preparation many preparation step application that runs eo of several preparation preparation pipeline of execution time from at about 15 minutes whe threads and 23GB of 1 reduces the runtime fri sequencing data for h computing time, and t Figures	elPrep (max RA	y coordina aad group ols ep ed M)	s fil	ter unmapper ter sequence		mark duplicate merged

ເກາຍດ

Go Gopher image by Renee French, CC BY 3.0, https://creativecommons.org/licenses/by/3.0/

PARGO

- Pargo available at
- Documentation:
- More documentation:
- elPrep:

https://github.com/exascience/pargo https://godoc.org/github.com/exascience/pargo https://github.com/exascience/pargo/wiki https://github.com/exascience/elprep



embracing a better life