



Master's Thesis Proposals

Professor Eric Jul
Programming and Software Engineering Group
ericbj@ifi.uio.no

Fall 2020

I teach IN5570 Distributed Objects & IN3030 Efficient Parallel Programming every Spring.
Veiledning: enten på **dansk** or **English (American)**

Strong believer in ***Learning-by-doing*** – so project are much about *programming*

Project areas:

- Object-oriented Programming Languages
- Distributed Objects and Cloud Programming
- Parallel Programming for Multi-Core systems – in cooperation with Arne Maus
- Design Patterns





Parallel Computing for Multi-Core Systems

- **Effektiv, cache-venlig Primtall generering og faktorisering**
I IN3030 er en av de obligatoriske oppgaver parallellisering av Primtall generering og faktorisering av store tal. I dette prosjekt vil algoritmen blive videreutviklet for at oppnå maksimal yteevne. Spesielt ut over en optimal parallellisering skal algoritmen gjøres cache-venlig. Mulighet for en artikkel i fortsettelse av tidligere artikkel (publiseret NIK 2020).
- Sammen med Arne Maus, Emeritus



Parallel Computing for Multi-Core Systems

- **Raskere generering av primtall for å bestemme store primtallsgap ved kombinerings av Eratosthenes sil og faktorisering.**

▪



Nytt Gap fra= 18 361 375 334 787 046 697, len= 1550

0 - 18 361 375 334 787 046 697 = 18 361 375 334 787 046 697 --PRIME--

2 - 18 361 375 334 787 046 699 = 19*1 499*5 615 587* 114 803 417

4 - 18 361 375 334 787 046 701 = 3 *16 290 299* 375 711 854 333

134 - 18 361 375 334 787 046 831 = 2 151 691 249*8 533 461 919

186 - 18 361 375 334 787 046 883 = 62 231 947* 295 047 418 889

594 - 18 361 375 334 787 047 291 = 491*37 395 876 445 594 801

596 - 18 361 375 334 787 047 293 = 5 692 007*3 225 817 419 899

598 - 18 361 375 334 787 047 295 = 3 * 5 *79*97*4 456 513*35 844 287

1288 - 18 361 375 334 787 047 985 = 3 * 3 * 5 *6 269*65 087 025 521 657

1290 - 18 361 375 334 787 047 987 = 1 237*1 622 359*9 149 314 289

1292 - 18 361 375 334 787 047 989 = 3 731 909*4 920 102 643 121

1294 - 18 361 375 334 787 047 991 = 3 *19* 136 920 761*2 352 670 183

1546 - 18 361 375 334 787 048 243 = 3 *79*4 549*7 307*2 330 782 873

1548 - 18 361 375 334 787 048 245 = 5 *11*61 717*1 105 691*4 892 197

1550 - 18 361 375 334 787 048 247 = 18 361 375 334 787 048 247 --PRIME--

Her har jeg tatt ut alle de tallene som har store første faktor. Bare ett er et problem (tall 186) + start og slutt (resten starter på små primtall som førstfaktor)

Ide: Ser vi på faktoriseringen av det største primgapet ($< 2^{64}$), så ser vi:
 1 De aller fleste første faktorer er små.
 2 Alle faktoriseringer som tilsier at vi har et ikke-primtall, tas fra de første primtallene $< 10^8$. Med ett unntak av tall 186 som har en førstefaktor på 62 231 947.

Algoritme:

a) Gå gjennom våre store d-segmenter med de 5,7 mill første primtallene med Eratosthenes Sil.

B) gå så gjennom d-segmentene og plukk ut de få primtallene som har mange ikke-primtall på begge sider av seg.

C) prøv så å faktorisere disse videre ved divisjon

Vi jobber med Sannsynlige Primtall og kombinerer Eratosthenes Sil med litt divisjonsfaktorisering.



Investigating the Near-Far Computing Model

The concept of Near-Far Computing has been proposed to mitigate the latency problems caused by the slow speed of light and the poor resources available at the edge of the cloud.

Latency due to the slow speed of light means that there is an inherent, inescapable latency when accessing remote data centers, so applications that require low latency **MUST** run near the edge of the cloud.

On the other hand, the resources available at the edge are limited.

The Near-Far Model introduces intermediate servers between edge devices and remote data centers.

This thesis will investigate the model by programming a specific example and showing the viability of using the model.

Part of the project is finding a suitable application – suggestions are Virtual Reality Games deployed in real life scenarios, distributed P2P systems, and caching systems.



Distributed Objects – Language Mechanisms

- **Implementing the TRACK mechanism in Emerald**
 - TRACK is a proposed language construct that allows an object, A, to keep track of where another object is currently located in a distributed system.
 - Combination of implementation (both compiler and runtime) and language level usage of the concept.
 - Co-author an article with me on the subject for a conference

- **Type-based Access Control, Locks, and Authentication**
 - Using the restrict-to mechanism to limit access control
 - Implement a number of examples of how to provide language based control mechanism
 - Co-author an article with me on the subject for a conference



Distributed Objects

- **Watch dog service:** Write an Emerald program that implements a watch dog service that runs on Planetlab. The service provides the ability to keep a user service running in a number of replicas by monitoring the replicas and starting up new replicas when previous ones disappear, e.g., due to node crashes.
- **Network map:** Write an Emerald program that maps out the internet nodes where your Emerald program runs. It is to replicate itself onto all available nodes and then monitor which nodes are up, how long they have been running, and what the round trip delays are between the nodes.



Languages for Concurrency and Distribution

- **Comparison of the Design of Java and Emerald.** Compare the design decisions made for these two languages. Use programs to illustrate strengths and weaknesses of each design decision.
- **Comparison of Concurrency Mechanisms for Emerald and Erlang.** Compare the design decisions made for these two languages. Use programs to illustrate strengths and weaknesses of each design decision.



Design Patterns

- **Design Patterns for Mobile Devices**
- **Design Patterns for Cloud Computing**

A comparative study of either area.

Must be combined with the actual implementation of multiple design patterns in a prototype system.

These are wide-open areas – there are several previous work, so you will have to focus on a particular subarea.