OC4MC

OCaml for MultiCore

Deuxième réunion du GDR GPL/LTP

LACL / Université Paris XII

Mathias Bourgoin, Adrien Jonquet
Benjamin Canou, Philippe Wang
Emmanuel Chailloux

21 Octobre 2009
Jane Street OCaml summer project

( pause )

Mathias' research project

broken implementations

numerous bug fixes

continuing
multithreading based multicore profit with OCaml

- (super light) preliminaries
- chapter 1 / OCaml & Parallelism
- chapter 2 / OCaml for MultiCore
- chapter 3 / Performance
- chapter 4 / Conclusion & Future Work
dependencies mess
interdependencies

if there is a garbage collector and there are threads, then their implementations are probably interdependent
chapter 1

OCaml & Parallelism
multitask programming

ASM
C
Java
Join Java
OCaml
JOCamll
MultiLisp
Clojure
assembly tweaking
SIMD GPU
NVIDIA GeForce GT130

cooperative threads
preemptive threads
(heavy) process

SML
Concurrent ML
Alice

CS
join calculus
concurrent dataflow
CUDAl
grid
p2p
Oz

cluster
internet computing

Haskell
Concurrent Haskell

PPE+SPEs
CELL

shared memory
message passing

MIMD CPU
Intel Core Quad

π-calculus
pict

(cluster)

(what a mess around)

(preemptive thread)

(preemptive thread)

(cooperative threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(cooperative threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)

(pre preemptive threads)
multitask programming

Concurrent Haskell  JOCaml  Concurrent ML
Alice  Join Java  \( \pi \)-calculus  pict  CCS
OCaml  Haskell  SML  Scala  Io  Oz
OC4MC  MultiLisp  Ada
Clojure  Java  join calculus  concurrent dataflow

cooperative threads  preemptive threads  (heavy) process
C  ASM  assembly tweaking

SIMD  GPU  PPE+SPEs  MIMD  CPU
NVIDIA GeForce GT130  CELL  Intel Core Quad
internet computing  p2p  cluster  grid
message passing  shared memory
some Caml history

1985 1991 1996 today

Caml Caml Light Objective Caml

cheap multicores
Objective Caml

- Functional-based multiparadigm language
- Distribution by INRIA
  - known for its efficiency
  - non-parallel concurrent threads
some parallel threads issues

- runtime library support
- reentrance (beware of shared static variables)
- memory allocation/collection
addressing some parallel threads issues

- runtime library support
  - reentrance (beware of shared static variables)
    - use (POSIX) `__thread` facility
  - transform to function parameters
  - refactoring
- memory allocation/collection
addressing some parallel threads issues

- runtime library support
  - reentrance (beware of shared static variables)
  - memory allocation/collection
    - look at the guts, be scared, run or make a choice
      - learn, understand, adapt
      - learn, remove parts & rewrite from scratch
memory management

- allocation
  - `heap_ptr = heap_ptr - size(val)`

- two-generation collection
  - minor collection: stop&copy
  - major collection: incremental mark&sweep&compact
INRIA OCAML
garbage collection

young heap

Stop&Copy
survivors
to old heap

old heap

Mark&Sweep
+ compact

(runtime lib implementation in C code + asm)

very efficient
generational
incremental
chapter 2

OCaml for MultiCore
(influence of the) past experience

Pagano et al. (JFLA 2007, PADL 2008, ICFP 2009)

- alternative runtime lib implementation for software certification required by civil avionics norms

- memory management too hard to explain thus impossible to certify

- remove concurrency, marshalling, weak pointers, ...

- result: from 16 000 lines of C code to 4 500
addressing some parallel threads issues

- runtime library support
  - reentrance (beware of shared static variables)
  - memory allocation/collection
    - look at the guts, be scared, run or make a choice
    - learn, understand, adapt
    - learn, remove parts & rewrite from scratch

and what about scheduling?
interdependencies

if the memory management implies moving values, then it can stop a thread from accessing values...
OCaml for MultiCore

- new memory management
- **stop** the world & two-generation **copy**
- “inherited” mechanisms
  - stop at allocation, blocking operations
- parallel threads
Partial Collection

OC4MC

Pages

Before

Thread 1

Thread 2

Thread 3

Shared heap

Partial GC

Stop & Copy

After
OC4MC

Full Collection

Stop&Copy

Before

Full GC

After

new shared heap

thread 1

thread 2

thread 3

pages
Execution Sample

- **allocation failure**
- **suspended allocation**
- **blocking operation**
- **allocation ok**
- **blocking operation**
- **thread is running normally**
- **thread is sleeping**
- **tick thread**

Full GC
Execution Sample

- allocation failure
- suspended allocation
- blocking operation
- allocation ok
- blocking operation
- thread is running normally
- thread is sleeping

OCaml For MultiCore
chapter 3

performance
## Speedups

<table>
<thead>
<tr>
<th></th>
<th>O’Caml</th>
<th>OC4MC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong># of threads</strong></td>
<td>1TH</td>
<td>1TH</td>
</tr>
<tr>
<td><strong>Sieve</strong></td>
<td>60s</td>
<td>64s</td>
</tr>
<tr>
<td>speedup</td>
<td>1.06</td>
<td>1</td>
</tr>
<tr>
<td><strong>Matmult</strong></td>
<td>15.5s</td>
<td>18.2s</td>
</tr>
<tr>
<td>speedup</td>
<td>1.17</td>
<td>1.92</td>
</tr>
<tr>
<td><strong>Life</strong></td>
<td>24.3s</td>
<td>24.7s</td>
</tr>
<tr>
<td>speedup</td>
<td>1.02</td>
<td>1.49</td>
</tr>
</tbody>
</table>

### little number of threads

<table>
<thead>
<tr>
<th></th>
<th>O’Caml</th>
<th>OC4MC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sieve CML-style</strong></td>
<td>89s</td>
<td>59s</td>
</tr>
<tr>
<td>speedup</td>
<td>1</td>
<td>1.50</td>
</tr>
</tbody>
</table>

### great number of threads

X86-64 Quad Core X 2
Virtual Machine : 7 active cores
Slowdowns

- GC algorithm kept as simple as possible
  - heap growth can slow down the program
  - very functional style (many short life objects) combined with some long life objects shows stop&copy weakness
- Predictable weakness
chapter 4

conclusion & future work
Conclusion

it’s working!

http://www.algo-prog.info/ocmc/
open source distribution

- Linux x86 64-bit
- Alternative multicore capable runtime library
  - memory manager replacement
  - thread library replacement
- Available as a patch for OCaml 3.10.2
  - http://www.algo-prog.info/ocmc/
Conclusion

- working multicore capable threads for OCaml
- proof of feasibility
- potential good performance

http://www.algo-prog.info/ocmc/
Related & Future Work

- OCaml concurrent/parallel extensions
  - what if they are used with OC4MC
- Use a limited number of threads by implementing abstractions on system threads
  - e.g. Parallel Concurrent ML
- Alternative GC algorithm

http://www.algo-prog.info/ocmc/