Quicksort, again

David Kitchin, Adrian Quark, Jayadev Misra

Department of Computer Science
University of Texas at Austin

http://orc.csres.utexas.edu

Describe Quicksort using:

- Recursion: nicely done in Functional programming
- Mutable Data Structure: nicely done in Imperative programming
- Concurrency: ?

Orc, an Orchestration Theory

- Site: Basic service (component).
- Concurrency combinators for integrating sites.
- Theory includes nothing other than the combinators.

  No notion of data type, thread, process, channel, synchronization, $\cdots$

Sites

- A site is called like a procedure with parameters.
- Site returns at most one value.
- The value is published.

Site calls are strict.

Examples of Sites

- $+$ $-$ $*$ $\&\&$ $||$ $<$ $=$ ...
- `println`, `random`, `Prompt`, `Email` ...
- `Ref`, `Semaphore`, `Channel`, `Database` ...
- `Timer`
- External Services: Google Search, MySpace, CNN, ...
- Any Java Class instance
- Sites that create sites: `MakeSemaphore`, `MakeChannel` ...

  ...

Overview of Orc

- Orc program has
  - a set of definitions,
  - a goal expression, over sites and combinators.
- The goal expression is executed. Its execution
  - calls sites,
  - publishes values.

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f >x> g$	Sequential composition
for some x from g do f	$f <x< g$	Pruning

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f > x > g$	Sequential composition
for some x from g do f	$f < x < g$	Pruning

Structure of Orc Expression

- Simple: just a site call, $CNN(d)$
  Publishes the value returned by the site.

- Composition of two Orc expressions:

do f and g in parallel	$f \mid g$	Symmetric composition
for all x from f do g	$f >x> g$	Sequential composition
for some x from g do f	$f <x< g$	Pruning

Examples

- $CNN(d) \mid BBC(d)$:
  Calls CNN and BBC simultaneously.
  Publishes up to two values.

- $(CNN(d) \mid BBC(d)) > x > Email(address, x)$:
  Publishes up to two values from $Email$.

- $Email(address, x) < x < (CNN(d) \mid BBC(d))$:
  Publishes at most one value from $Email$.

Some Fundamental Sites

- $if(b)$: boolean b,
  returns a signal if b is true; remains silent if b is false.
- $Rtimer(t)$: integer t, $t \geq 0$, returns a signal t time units later.
- $stop$: never responds. Same as $if(false)$.
- $signal$: returns a signal immediately. Same as $if(true)$.

Publish a signal at every time unit.

$$def \;\; metronome() = signal \mid (Rtimer(1) \gg metronome())$$

Laws Based on Kleene Algebra

(Zero and $\mid$)	$f \mid stop = f$
(Commutativity of $\mid$)	$f \mid g = g \mid f$
(Associativity of $\mid$)	$(f \mid g) \mid h = f \mid (g \mid h)$
(Idempotence of $\mid$) NO	$f \mid f = f$
(Associativity of $\gg$)	$(f \gg g) \gg h = f \gg (g \gg h)$
(Left zero of $\gg$)	$stop \gg f = stop$
(Right zero of $\gg$) NO	$f \gg stop = stop$
(Left unit of $\gg$)	$Signal \gg f = f$
(Right unit of $\gg$)	$f > x > let(x) = f$
(Left Distributivity of $\gg$ over $\mid$) NO	$f \gg (g \mid h) = (f \gg g) \mid (f \gg h)$
(Right Distributivity of $\gg$ over $\mid$)	$(f \mid g) \gg h = (f \gg h \mid g \gg h)$

Additional Laws

(Distributivity over $\gg$) if g is x-free

$$((f \gg g) <x< h) = (f <x< h) \gg g$$

(Distributivity over $|$) if g is x-free

$$((f \mid g) <x< h) = (f <x< h) \mid g$$

(Distributivity over $<<$) if g is y-free

$$\begin{array}{rl} & ((f <x< g) <y< h) \\ = & ((f <y< h) <x< g) \end{array}$$

(Elimination of where) if f is x-free, for site M

$$(f <x< M) = f \mid (M \gg stop)$$

Operational semantics of Orc

- Traces of an expression.
  Special treatment for unbound variables.
- Ordering of expressions based on subset ordering over traces.
  Congruence.
- Combinators are monotonic and continuous.
- Holds in the timed model too.

Encoding

To compute $1 + 2 + 3$:

$$add(1, x) \; < x < \; add(2, 3)$$

Functional Core Language

- Data Types: Number, Boolean, String, with usual operators
- Conditional Expression: if E then F else G
- Data structures: Tuple and List
- Pattern Matching
- Function Definition; Closure

Variable Binding; Silent expression

val $x = 1 + 2$
val $y = x + x$
val $z = x/0$ - - expression is silent
val $u =$ if $(0 < 5)$ then 0 else z

Example: Fibonacci numbers

def $H(0) = (1, 1)$
def $H(n) =$
  val $(x, y) = H(n-1)$
  $(y, x+y)$

def $Fib(n) =$
  val $(x, -) = H(n)$
  x

– Goal expression
$Fib(5)$

Translating Functional Core to Pure Orc

- Operators to Site calls:
  $1 + (2 + 3)$ to $add(1, x) \;\; <x< \; add(2, 3)$
- if E then F else G:
  $(if(b) \gg F \mid not(b) > c > if(c) \gg G) \;\; <b< \; E$
- val $x = G$ followed by F:
  $F \;\; <x< \; G$
- Data Structures, Patterns: Site calls and variable bindings
- Function Definitions: Orc definitions

Comingling with Orc expressions

Components of Orc expression could be functional.
Components of functional expression could be Orc.

$1+2 \mid 2+3$,
is $((let(x) \mid let(y)) < x < add(1,2)) < y < add(2,3)$

Convention: whenever expression F appears in a context where a single value is expected, convert it to $x < x < F$.

$(1 \mid 2) + (2 \mid 3)$,
is $(add(x,y) < x < (1 \mid 2)) < y < (2 \mid 3)$

Example: Fibonacci numbers

def $H(0) = (1, 1)$
def $H(n) = H(n-1)$ $>(x, y)>$ $(y, x+y)$

def $Fib(n) = H(n)$ $>(x, -)>$ x

– Goal expression
$Fib(5)$

Dice Throw

$$def\ \ throw() = \ random(6) + 1$$

$$
\begin{aligned}
&def\ \ exp(0) = \ 0\\
&def\ \ exp(n) = \\
&\qquad (\text{if }\ throw() + throw() = 7 \text{ then }\ 1 \text{ else }\ 0)\\
&\quad + exp(n-1)
\end{aligned}
$$

Dice Throw translated

$$\mathit{def}\ \ \mathit{throw}() = \ \mathit{add}(x,1) < x < \mathit{random}(6)$$

$$
\begin{array}{l}
\mathit{def}\ \ \mathit{exp}(n) = \\
\quad (\ \mathit{if}(b) \gg \mathit{let}(0) \\
\qquad \mid \mathit{not}(b)\ \ {>}nb{>}\ \mathit{if}(nb) \\
\qquad (\ \ \mathit{add}(x,y) \\
\qquad\qquad {<}x{<}\ ((\mathit{if}(bb) \gg 1\ \mid \mathit{not}(bb)\ \ {>}nbb{>}\ \mathit{if}(nbb) \gg 0) \\
\qquad\qquad {<}bb{<}\ \mathit{equals}(p,7) \\
\qquad\qquad\quad {<}p{<}\ \mathit{add}(q,r) \\
\qquad\qquad\qquad {<}q{<}\ \mathit{throw}() \\
\qquad\qquad\qquad {<}r{<}\ \mathit{throw}()) \\
\qquad\qquad {<}y{<}\ (\mathit{exp}(m)\ \ {<}m{<}\ \mathit{sub}(n,1)) \\
\qquad) \\
\quad)\ \ \ {<}b{<}\ \mathit{equals}(n,0)
\end{array}
$$

Fork-join parallelism

Call sites M and N concurrently.
Return their values as a tuple after both respond.

$$((u, v) \; < u < \; M()) \; < v < \; N()$$, or simply

$$(M(), N())$$

Mutable Structures

$\text{val}\ \ r = \mathit{Ref}()$

$r.\mathit{write}(3)$, or $r := 3$
$r.\mathit{read}()$, or $r?$

$$\mathit{def}\ \ \mathit{swapRefs}(x, y) = x? \;>z>\; x := y? \gg y := z$$

Random Permutation

$val \;\; N = \;\; 20$ - - size of permutation array
$val \;\; ar = \;\; fillArray(Array(N), lambda(i) \; = i)$

- - Randomize array a of size n, $n \geq 1$
$def \;\; randomize(1) = \;\; signal$
$def \;\; randomize(n) =$
$\quad random(n) \;\; >k>$
$\quad swapRefs(ar(n-1), ar(k)) \;\; \gg \;\; randomize(n-1)$

$randomize(N)$

Binary Search Tree; Pointer Manipulation

$def \;\; search(key) = \;\;$ -- return true or false

$\quad searchstart(key) \;\; >(-,-,q)> \; (q \neq null)$

$def \;\; insert(key) = \;\;$ -- true if value was inserted, false if it was there

$\quad searchstart(key) \;\; >(p,d,q)>$

$\quad$ if $\; q = null$

$\qquad$ then $\; Ref() \;\; >r>$

$\qquad\quad r := (key, null, null) \gg update(p,d,r) \gg true$

$\qquad$ else $\; false$

$def \;\; delete(key) =$

Semaphore

$val\ \ s = \ Semaphore(2)$ - - s is a semaphore with initial value 2

$s.acquire()$
$s.release()$

Rendezvous:

$val\ \ s = \ Semaphore(0)$
$val\ \ t = \ Semaphore(0)$

$def\ \ send = \quad t.release() \gg s.acquire()$
$def\ \ receive = t.acquire() \gg s.release()$

n-party Rendezvous using $2(n-1)$ semaphores.

Readers-Writers

$$
\begin{array}{l}
val\ \ req = \ Buffer() \\
val\ \ cb = \ Counter() \\
\\
def\ \ rw() = \\
\quad req.get()\ \ {>}(b,s){>} \\
\qquad (\ \ if(b)\ \gg\ cb.inc()\ \gg\ s.release()\ \gg\ rw() \\
\qquad \ \ |\ if(\neg b)\ \gg\ cb.onZero() \\
\qquad \qquad \gg\ cb.inc()\ \gg\ s.release()\ \gg\ cb.onZero()\ \gg\ rw() \\
\qquad) \\
\\
def\ \ start(b) = \\
\quad val\ \ s = \ Semaphore(0) \\
\quad req.put((b,s))\ \gg\ s.acquire() \\
\\
def\ \ quit() = \ cb.dec()
\end{array}
$$

Scan, swap over array a

$$def\ \ lr(i) = \ \text{if } (i < t \ \wedge\ a(i)? \leq p) \text{ then } lr(i+1) \text{ else } i$$

$$def\ \ rl(i) = \ \text{if } (a(i)? > p) \text{ then } rl(i-1) \text{ else } i$$

$$def\ \ swap(i,j) = \ a(i)?\ \ {>}z{>}\ a(i) := a(j)?\ \gg\ a(j) := z$$

Partition

$$
\begin{aligned}
&\mathit{def}\ \ \mathit{part}(p,s,t) = \\
&\quad \mathit{def}\ \ \mathit{lr}(i) = \ \text{if } (i < t\ \wedge\ a(i)? \le p) \text{ then } \mathit{lr}(i+1) \text{ else } i \\
&\quad \mathit{def}\ \ \mathit{rl}(i) = \ \text{if } (a(i)? > p) \text{ then } \mathit{rl}(i-1) \text{ else } i
\end{aligned}
$$

$$(\mathit{lr}(s), \mathit{rl}(t)) \ \ {>}(s', t'){>}$$

$$(\ \ \mathit{if}(s'+1 < t') \gg \mathit{swap}(s', t') \gg \mathit{part}(p, s'+1, t'-1)$$

$$\mid \mathit{if}(s'+1 = t') \gg \mathit{swap}(s', t') \gg s'$$

$$\mid \mathit{if}(s'+1 > t') \gg t'$$

$$)$$

Returns m where

$$
\begin{aligned}
&a(s)\cdots a(m) \le p, \\
&a(m+1)\cdots a(t) > p
\end{aligned}
$$

Sorting

$$
\begin{aligned}
&\mathit{def}\ \ \mathit{sort}(s,t) = \\
&\quad \text{if}\ \ s \geq t\ \text{then}\ \ \mathit{signal} \\
&\quad \text{else}\ \ \mathit{part}(a(s)?, s+1, t)\ \ >m> \\
&\qquad\quad \mathit{swap}(m,s)\ \ \gg \\
&\qquad\quad (\mathit{sort}(s, m-1), \mathit{sort}(m+1, t))\ \ \gg \\
&\qquad\quad \mathit{signal}
\end{aligned}
$$

$$\mathit{sort}(0, a.\mathit{length}() - 1)$$

Putting the Pieces together

$$
\begin{array}{l}
def\ \ quicksort(a) = \\
\quad def\ \ swap(x,y) = \ a(x)?\ >z>\ a(x) := a(y)?\ \gg\ a(y) := z \\
\quad def\ \ part(p,s,t) = \\
\qquad def\ \ lr(i) = \ \ \text{if } (i < t\ \wedge\ a(i)? \leq p) \text{ then } lr(i+1) \text{ else } i \\
\qquad def\ \ rl(i) = \ \ \text{if } (a(i)? > p) \text{ then } rl(i-1) \text{ else } i \\
\qquad (lr(s), rl(t))\ \ >(s',t')> \\
\qquad (\ \ if(s'+1 < t')\ \gg\ swap(s',t')\ \gg\ part(p, s'+1, t'-1) \\
\qquad\ \ |\ if(s'+1 = t')\ \gg\ swap(s',t')\ \gg\ s' \\
\qquad\ \ |\ if(s'+1 > t')\ \gg\ t' \\
\qquad) \\
\quad def\ \ sort(s,t) = \\
\qquad \text{if }\ s \geq t \text{ then }\ signal \\
\qquad \text{else }\ \ part(a(s)?, s+1, t)\ \ >m> \\
\qquad\qquad\quad swap(m,s)\ \gg \\
\qquad\qquad\quad (sort(s, m-1), sort(m+1, t))\ \gg \\
\qquad\qquad\quad signal \\
sort(0, a.length() - 1)
\end{array}
$$

Remarks and Proof outline

- Concurrency without locks
- $sort(m, n)$ sorts the segment; does not touch items outside the segment.
- Then, $sort(s, m-1)$ and $sort(m+1, t)$ are non-interfering.
- $part(p, s, t)$ does not modify any value outside this segment. May read values.

Acknowledgement

Tony was/is a major inspiration in the development of Orc.