Exploring the possibilities and limitations of Concurrent Programming for Multimedia Interaction and Visual Programming for Musical CSP’s

Mauricio Toro Bermúdez - mauriciotorob at gmail.com

IRCAM / Pontificia Universidad Javeriana de Cali
Supervised by Gérard Assayag – Carlos Agón / Camilo Rueda

Internship Report - December 18 2008
Outline

- Lightweight threads for Common Lisp
- Interpreters for CCP
- Ntccrt: A real-time capable interpreter for ntcc
- Gelisp: A new visual programming library to solve CSP's
- Results & Conclusions
Using Continuation Passing Style (CPS)

Figure: Traversing a List of $n$ elements. CSP done by using cl-cont.
Event-driven Programming

; Lightweight threads for Common Lisp using Event-driven Programming

; Thread data structures
(defvar *Runnable* nil)
(defvar *Current-thread* nil)
(defvar *Suspended-lock* (make-hash-table))
(defvar *Lett-variables* (make-hash-table))
(defstruct thread name status whoami waitingfor EventQueue Priority)

; Thread functions
(defun init () ...)
(defun thread-run-function (f parameters priority name) ...)

; Events
(defstruct execute body)
(defstruct bind who)
(defstruct wait who)
(defstruct lets declarations body)
(defstruct waitlock who body)
(defstruct dotimes varname init end step body)
(defstruct letss body)

; Event handlers
(defun executeHandler (body) ...)
(defun letshandler (body) ...)
(defun dotimesHandler (varname value end step body) ...)
(defun waitHandler (who body) ...)
Comparing our implementation with existing ones

**Figure:** Multithreaded matrix multiplication (time in seconds)
Future Work: Use CMU-CL

Figure: CMUCL: a high-performance, free Common Lisp implementation
Process interaction in CCP (1)

- `tell pitch2 > 60`
- `ask pitch1 > 58 do P`
- `tell pitch1 > pitch2 + 2`
- `ask pitch1 = 58 do Q`

STORE

Mauricio Toro Bermúdez - mauriciotorob at gmail.com

Exploring the possibilities and limitations of Concurrent Programming
Process interaction in CCP (2)

pitch2 > 60
STORE
pitch1 > pitch2 + 2

ask pitch1 = 58 do Q
Gecode

Figure: Generic Constraint Development Environment

Mauricio Toro Bermúdez - mauriciotorob at gmail.com
Exploring the possibilities and limitations of Concurrent Programming
Threaded Interpreters in Lisp and C++

- a) Using LispWorks processes
- b) Using Pthreads in C++
Using Event-driven interpreter in Lisp

- Lightweight threads for Common Lisp
- CCP Interpreter
- Ntccrt
- Gelisp
- Conclusions

Concurrent Constraint Programming
Implementation using threads
Implementation using E.D. in Lisp

Using Event-driven interpreter in Lisp

- PROCESS 1
- PROCESS 2
- PROCESS N

EVENTS

MAIN LOOP

SCHEDULER

DISPATCHER

TELL HANDLER

ASK HANDLER

PARALLEL HANDLER

GECOL

GECODE STORE

LISP

C++

Mauricio Toro Bermúdez - mauriciotorob at gmail.com

Exploring the possibilities and limitations of Concurrent Programs
There is a dispute between computer scientists about the development of multimedia interaction systems (e.g., music improvisation).

A group argue that in order to implement real-time capable systems, those systems should be written directly in C++ for efficiency.

The other group argue that those systems should not be written directly in C++; they should be modeled using a formalism and execute those models on a real-time capable interpreter.
Developing programs with Ntccrt

User

Common Lisp

OpenMusic

Ntccrt

Pure Data

Max/Msp

Stand-alone

Programmer

C++
Unless agent in \texttt{ntcc}

\begin{itemize}
  \item Unless pitch1 = 60 next
  \item unless pitch1 = 60 next
  \item unless pitch1 = 60 next
\end{itemize}

\textbf{a)} There is not information about pitch1

\textbf{b)} pitch1 is equal to 61

\textbf{c)} pitch1 is equal to 60
Non-deterministic agent in **ntcc**

- **STORE**
  - `pitch1 = 52`
  - `tell (pitch1 = 48) + tell (pitch1 = 52) + tell (pitch1 = 55)`
  - **TIME UNIT = 0**

- **STORE**
  - `pitch1 = 48`
  - `tell (pitch1 = 48) + tell (pitch1 = 52) + tell (pitch1 = 55)`
  - **TIME UNIT = 2**

- **STORE**
  - `pitch1 = 55`
  - `tell (pitch1 = 48) + tell (pitch1 = 52) + tell (pitch1 = 55)`
  - **TIME UNIT = 3**

- **STORE**
  - `pitch1 = 55`
  - `tell (pitch1 = 48) + tell (pitch1 = 52) + tell (pitch1 = 55)`
  - **TIME UNIT = 1**
Example: CCFOMI

The $\text{SYNC}_i$ process in $\text{ntcc}$

$\text{SYNC}_i \overset{def}{=} \begin{array}{l}
\text{when } S_{i-1} \geq -1 \land go \geq i \text{ do } \\
(ADD_i \parallel \text{next } \text{SYNC}_{i+1}) \\
\text{unless } S_{i-1} \geq -1 \land go \geq i \text{ next } \text{SYNC}_i)
\end{array}$
Figure: Writing the $\textit{SYNC}_i$ process in OpenMusic
The \textit{SYNC}_i process in Common Lisp

\begin{verbatim}
(defproc sync (i)
  (whenp (andc (v<= go i) (v<= (ntccaref S (v- i 1)) -1))
    (parallelp (nextnp (callp sync (v+ i 1)))
     (callp add i))
  )
  (unlessp (andc (v<= go i) (v<= (ntccaref S (v- i 1)) -1))
     (nextnp sync i)
  )
)
\end{verbatim}
The \textit{SYNC}; process in C++

```cpp
Gecode::Int::AskBody * synccp::Execute()(  
    Space * h,  
    vector<int> intparameters,  
    vector<variable *> variableparameters )  
{
    int i = intparameters[0];
    return parallelp(  
        whenp(ANDc(GQc(store->ref_IntV(S,i-1,h), -1),  
                  GQc(go,i)), parallelp(callp(Add,i),  
                         nextnp(callp(Sync,i+1)))),  
        unlessp(ANDc(GQc(store->ref_IntV(S,i-1,h), -1),  
                  GQc(go,i)), nextnp(callp(Sync,i))) );
}
```

Mauricio Toro Bermúdez - mauriciotorob at gmail.com

Exploring the possibilities and limitations of Concurrent Programming for Multimedia Interaction and Visual Programming for Musical CSP's
Non-deterministic Timed Concurrent Constraint
Example: Synchronizing an improvisation system
Implementation using Gecode propagators
Future work: Using rtcc
Future work: Using pntcc
Future Work: Another Graphical representation

Figure: Running CCFOMI in Pure Data (PD)
Results

- We ran CCFOMI in Ntccrt over an Intel 2.8 GHz using Mac OS 10.5.2 and GCC 4.1, taking an average of 20 milliseconds per time-unit, scheduling around 880 processes per time-unit, and simulating 300 time-units.

- “John McLaughlin, said to be one of the fastest Jazz guitarists, and found a minimum inter onset time of about 60 milliseconds. This figure gives an approximate constraint for the computation time of our system: it should be able to learn and produce sequences in less than 30 milliseconds.” [The continuator]
Constraints as Propagation Agents

Action of the propagator \( \text{pitch}_1 > \text{pitch}_2 + 2 \)

\[
\text{pitch}_1 \in [36..72] \\
\text{pitch}_2 \in [60..80] \\
\text{pitch}_1 > \text{pitch}_2 + 2 \\
\text{pitch}_1 \in [63..72] \\
\text{pitch}_2 \in [60..69]
\]

Non-deterministic Timed Concurrent Constraint
Example: Synchronizing an improvisation system
Implementation using Gecode propagators
Future work: Using rtcc
Future work: Using pntcc
Future work: Another Graphical representation
Lightweight threads for Common Lisp
CCP Interpreter
Ntccrt
Gelisp
Conclusions

Non-deterministic Timed Concurrent Constraint
Example: Synchronizing an improvisation system
Implementation using Gecode propagators
Future work: Using rtcc
Future work: Using pntcc
Future Work: Another Graphical representation

Encoding when processes as Gecode propagators

when a=c do P

STORE

Mauricio Toro Bermúdez - mauriciotorob at gmail.com

Exploring the possibilities and limitations of Concurrent Programs
Lightweight threads for Common Lisp
CCP Interpreter
Ntccrt
Gelisp
Conclusions


Encoding when processes as Gecode propagators

\[
\text{a} = \text{c} \leftrightarrow \text{b}
\]

\[
\text{if b then P else skip}
\]

STORE
Non-deterministic Timed Concurrent Constraint

Example: Synchronizing an improvisation system

Implementation using Gecode propagators

Future work: Using rtcc

Future work: Using pntcc

Future Work: Another Graphical representation

Encoding $\sum$ processes as Gecode *propagators*

$$\sum_{x \in P} \text{when } x \in \text{wait}_j \text{ do ( tell (control}_j = x))$$

Mauricio Toro Bermúdez - mauriciotorob at gmail.com

Exploring the possibilities and limitations of Concurrent Programming
Encoding $\sum$ processes as Gecode propagators

$1 \in \text{wait}_j \leftrightarrow b_1$

$2 \in \text{wait}_j \leftrightarrow b_2$

$\text{control}_j = min(\{1 < i < n, b_i = true\})$

$\text{STORE} \ b_1 \ b_2$
Lightweight threads for Common Lisp
CCP Interpreter
Ntccrt
Gelisp
Conclusions

Non-deterministic Timed Concurrent Constraint
Example: Synchronizing an improvisation system
Implementation using Gecode propagators
Future work: Using rtcc
Future work: Using pntcc
Future Work: Another Graphical representation

Real Time Concurrent Constraint (1)

Figure: Reactive system in the ntcc
Real Time Concurrent Constraint (2)

Figure: Reactive system in Ntccrt
Real Time Concurrent Constraint (3)

Figure: Reactive system in rtcc
Probabilistic \( \text{ntcc} \)

- “The uncertainty underlying concurrent interactions in areas such as computer music goes way beyond of what can be modeled using partial information only” [pntcc]
- \( \text{pntcc} \) overcomes that problem. It adds a new agent \( \bigoplus \) to \( \text{ntcc} \) for **probabilistic choice**.

A process choosing a *factor link* from the \( \text{FO} \) with a probability distribution \( \rho \).

\[
\bigoplus \quad \text{when } \sigma \in \text{from}_k \text{ do } (\text{tell}(\text{output} = \sigma), \rho_{\sigma})
\]

- Prism could be used for model checking.
Musical sketch using Pure Data

Figure: The colored figures represent data structures in C
All-interval Series

Figure: Finding solutions
All-interval Series

Figure: Defining the CSP
Future work: All-triad Hexachords

Figure: Finding All-interval series containing an Hexachord
Figure: Setting up the first and the last note
Figure: Setting up the harmony
Figure: Setting up the motives occurrence
Future work: Optimization based on the intervals
; A version using the standard "distinct" constraint
;; By Camilo Rueda, IRCAM and Universidad Javeriana-Cali 20/06/2006
(defun script-allintervals (n)
  (let* ((sp (make-instance 'ge-space))
         (notes (add-fd-variables sp n 0 (- n 1)))
         (dists (add-fd-variables sp (- n 1) (- 1 n) (- n 1)))
         (d (add-fd-variables sp (- n 1) 1 (- n 1)))
         ; all notes must be different
         (g-distinct notes)
         ; all intervals should be different
         (g-distinct d)
         ; just to break symetries
         (<g (nth 0 notes) (nth 1 notes))
         (<g (nth 0 dists) (nth (- n 2) dists))
         (loop for i from 0 to (- n 2) do
            (notes(i+1) - notes(i) = dists(i))
            (=g (-g (nth (+ i 1) notes) (nth i notes)) (nth i dists))
            (g-obs (nth i dists) (nth i d))
            (<g (nth 0 d) (nth (- n 2) d))
            (g-explore notes )
            sp))
  (setq myspace (script-allintervals 12))
  (time (search-next myspace))
  (release-space myspace)

**Figure:** Finding solutions using Common Lisp
Lightweight threads for Common Lisp
CCP Interpreter
Ntccrt
Gelisp
Conclusions

OpenMusic Interface
Common Lisp Interface

Michael Jarrell CSP

; limits (first and last note)
(defun (not (equal limits nil))
  (limit-scenario
    (if (first (add-fd-variables up 1 -1 11))
        (if (first notes) (>= (kg 12 X) (first limits)))
        (if (first notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (second limits))))

; interval occurrences
(defun for-occurrence-from-intervals (intervals)
  (limit-flows (loop for x in cell
    (loop for cell in intervals
      (loop for x in cell
        (loop for y in cell
          (loop for cell in intervals
            (loop for x in cell
              (loop for y in cell
                (loop for z in cell
                  (loop for w in cell
                    (cons x y z w))))))))))

(defun (not (equal intervals nil))
  (limit-scenario
    (if (first (add-fd-variables up 1 -1 11))
        (if (first notes) (>= (kg 12 X) (first limits)))
        (if (first notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (second limits))))

; loop for interval-index from 0 to intervals-size-1 do
(defun for-interval-index (cell interval-index)
  (limit-flows (loop for x in cell
    (loop for y in cell
      (loop for z in cell
        (loop for w in cell
          (cons x y z w))))))

(defun (not (equal cell nil))
  (limit-scenario
    (if (first (add-fd-variables up 1 -1 11))
        (if (first notes) (>= (kg 12 X) (first limits)))
        (if (first notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (second limits))))

; loop for interval-index from 0 to intervals-size-1 do
(defun for-interval-index (cell interval-index)
  (limit-flows (loop for x in cell
    (loop for y in cell
      (loop for z in cell
        (loop for w in cell
          (cons x y z w))))))

(defun (not (equal cell nil))
  (limit-scenario
    (if (first (add-fd-variables up 1 -1 11))
        (if (first notes) (>= (kg 12 X) (first limits)))
        (if (first notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (second limits))))

; Cache and constraints representing the occurrence of each cell in the intervals vector
(defun for-interval-index (cell interval-index)
  (limit-flows (loop for x in cell
    (loop for y in cell
      (loop for z in cell
        (loop for w in cell
          (cons x y z w))))))

(defun (not (equal cell nil))
  (limit-scenario
    (if (first (add-fd-variables up 1 -1 11))
        (if (first notes) (>= (kg 12 X) (first limits)))
        (if (first notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (second limits))))

; Notes belong to the chords or not
(defun for-interval-index (cell interval-index)
  (limit-flows (loop for x in cell
    (loop for y in cell
      (loop for z in cell
        (loop for w in cell
          (cons x y z w))))))

(defun (not (equal cell nil))
  (limit-scenario
    (if (first (add-fd-variables up 1 -1 11))
        (if (first notes) (>= (kg 12 X) (first limits)))
        (if (first notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (>= (kg 12 X) (second limits)))
        (if (first last notes) (second limits))))

; Figure: Defining Michael Jarrell CSP in Common Lisp
Conclusions (1)

- There are different models for lightweight concurrency. Depending on the application to run on top, different implementation may be required.

- Although Gecode was design to be a library for solving combinatorial problems using constraints, we found out that using Gecode for ntcc give us outstanding results for real-time.

- Most propagators used in real-time, have a reified version and those who do not have one, are easily extensible.
There are different models for lightweight concurrency. Depending on the application to run on top, different implementation may be required.

Although Gecode was design to be a library for solving combinatory problems using constraints, we found out that using Gecode for ntcc give us outstanding results for real-time.

Most propagators used in real-time, have a reified version and those who do not have one, are easily extensible.
Conclusions (1)

- There are different models for lightweight concurrency. Depending on the application to run on top, different implementation may be required.

- Although Gecode was design to be a library for solving combinatory problems using constraints, we found out that using Gecode for ntcc give us outstanding results for real-time.

- Most propagators used in real-time, have a reified version and those who do not have one, are easily extensible.
Conclusions (2)

- Using Gelisp we can solve CSP’s almost as fast as Gecode but representing heuristics and constraints using graphical boxes in an intuitive way.

- For this, we made all the tests under Mac OS X. Since Gecode is portable to other Linux and Windows, it will be easy to extend our applications for those OS’s in the future.
Conclusions (2)

- Using Gelisp we can solve CSP’s almost as fast as Gecode but representing heuristics and constraints using graphical boxes in an intuitive way.

- For this, we made all the tests under Mac OS X. Since Gecode is portable to other Linux and Windows, it will be easy to extend our applications for those OS’s in the future.
Willing to know more?

- A research report including the details of this work is available at the “Médiathèque de l’Ircam”.
- Ntccrt can be found at: http://ntccrt.sourceforge.net/
- Gelisp can be found at: http://gelisp.sourceforge.net/
M E R C I !