Hydra Parallel Programming System

Stefan Zwicklbauer

July 1, 2009

Inhaltsverzeichnis

Inhaltsverzeichnis

Aller Anfang war schwer

- Paralle Software am Anfang selten und unwichtig
- Intel und VIA sorgten für Durchbruch in Sachen Hardware
- Entwicklung von neuen Programmiersprachen bzw. Erweiterungen mit vereinfachter Parallelisierung
- Kompliziertere Software erfordert komplexere Implementationen

Es gibt verschiedene Ansätze um möglichst komfortabel, performanten Code zu entwickeln

Warum eine neue Sprache?

Warum benötigt man noch eine neue Sprache? Es gibt ohnehin Genug welche Parallelität beherrschen!

- Threadimplementierung kann sehr komplex sein
- Deadlocks/Livelocks stellen Problem dar
- JOMP wird nicht mehr weiterentwickelt, Jackal unvollständige OpenMP Implementierung
- Man wollte Erweiterung welche Synchronisierungsprobleme auf programmiertechnischer Ebene verhindert

Somit programmierte Franklin E. Powers, Jr eine Javaextension mit dem Namen HydraPPS ohne diese Probleme

Eigenschaften von Hydra

Allgemeine Eigenschaften von Hydra

- Abwärtskompatibilität
- keine Bytecodemodifikationen
- lauffähig mit allen relevanten Compilern, IDE's und Laufzeitumgebungen
- Jeglicher Programmcode wird atomar ausgeführt

Events

3 essenzielle Spracherweiterungen: **Events**, Eventgruppen, Locks

- Events statt Methoden
- Ausführung basiert auf Timeslot Mechanismus
- Anordnung der Events in sog. Eventgruppen
- Kennzeichnung durch @Event Annotation

Beispiel:

```
public @event void DoSomeWork() {
//Let's do some work
}
```

Timeslotmechanismus bei Events

3 essenzielle Spracherweiterungen: **Events**, Eventgruppen, Locks

- Hydra Laufzeitumgebung bestimmt den Ausführungszeitpunkt aufgrund internen Zeitleiste
- Events können nur bei gleicher Timeslotposition parallel ausgeführt werden
- Programmierer hat nur bedingt Einfluss darauf wann welcher Events wirklich ausgeführt wird
- 3 verschiedene Arten von Timeslots

Timeslotmechanismus bei Events

3 essenzielle Spracherweiterungen: **Events**, Eventgruppen, Locks

1. Logical time slot: Durch den Einsatz von Eventgruppen und Lockobjekten entscheidet Hydra auf welchen Timeslot dieser Event gesetzt wird. Kein Einfluss von Programmierer
2. Pre-scheduled event group time slot: Durch das Verwenden der Pre-Scheduled Eventgruppe kann aktiv Einfluss auf den Timeslot Mechanismus genommen werden
3. Runtime time slot: Bestimmt die tatsächliche Ausführungszeit der Events aufgrund eines "Mappings" der anderen beiden Timeslot Mechanismen

Eventgruppen

3 essenzielle Spracherweiterungen: Events, **Eventgruppen**, Locks

- Wird unter Angabe des Typs wie ein normales Objekt erstellt
- Bestimmen in welcher Reihenfolge platzierte Events ausgeführt werden sollen
- Komplexe Programmabläufe lassen sich durch Verschachtelungen von Eventgruppen erstellen
- 4 verschiedene Typen von Eventgruppen welche unterschiedliche "scheduling Strategien" erlauben

Problem: Programmierer macht sich weniger Gedanken um Lockobjekt, aber muss durch Abhängigkeitsanalyse eine sinnvolle Verschachtelung von Eventgruppen bestimmen

Eventgruppen

3 essenzielle Spracherweiterungen: Events, **Eventgruppen**, Locks

1. Sequenzielle Eventgruppe: Standardtyp - Events werden in einfacher sequenzieller Folge ausgeführt
2. FIFO Eventgruppe: Events werden in der Reihenfolge ausgeführt in der sie auch geschedult werden
3. Parallele Eventgruppe: Alle Events können parallel ausgeführt werden wenn die Locks der Parameter dies erlauben
4. Pre-scheduled: Mit "Forward" und "Backward" kann man Events selbst nach vorne und hinten verschieben. Kombination zwischen FIFO und Parallel.

Locks

3 essenzielle Spracherweiterungen: Events, Eventgruppen, **Locks**

- Es können nur Eventparameter gelockt werden
- Vor der Eventausführung werden Parameter zwischengespeichert
- Während der Ausführung können Locks nicht geändert werden und sind erst am Ende wieder frei verfügbar
- Eventausführung hängt von Lockfreigabe ab. Erst möglich wenn alle Parameter verfügbar sind

Mit Locks kann man Ausführungszeitpunkte setzen. Jedoch nicht empfehlenswert, da mehr Aufwand von Laufzeitumgebung vonnöten ist

Locks

3 essenzielle Spracherweiterungen: Events, Eventgruppen, **Locks**
Es gibt 4 verschiedene Locktypen:

1. pass: Event darf Parameter verwenden
2. read: Event benötigt Lesezugriff auf diesen Parameter
3. write: Event benötigt Schreibzugriff auf diesen Parameter
4. readwrite: Event benötigt Lese und Schreibzugriff

Je nachdem welche Locks verwendet werden, werden bei parallelen Eventgruppen die Events auf den Timeslots organisiert

Locks

3 essenzielle Spracherweiterungen: Events, Eventgruppen, **Locks**
Folgender Algorithmus wird verwendet ob ein Event auf dem gleichen Timeslot wie ein Anderer gesetzt werden kann:

```
ScheduleFor = 0

FOREACH Object IN ParametersOf(Event)
    IF PreviousLockFor(Object) == READ THEN
        IF RequestedLockFor(Object) == READ THEN
            Available = PreviousTimeSlotFor(Object)
        ELSEIF RequestedLockFor(Object) == WRITE THEN
            Available = PreviousTimeSlotFor(Object) + 1
        ELSEIF RequestedLockFor(Object) == READWRITE THEN
            Available = PreviousTimeSlotFor(Object) + 1
        ENDIF
    ELSEIF PreviousLockFor(Object) == WRITE THEN
        Available = PreviousTimeSlotFor(Object) + 1
    ELSEIF PreviousLockFor(Object) == READWRITE THEN
        Available = PreviousTimeSlotFor(Object) + 1
    ENDIF
    IF Available > ScheduleFor THEN
        ScheduleFor = Available
    ENDIF
NEXT Object

FOREACH Object IN ParametersOf(Event)
    SetPreviousLockFor(Object, RequestedLockFor(Object))
    SetPreviousTimeSlotFor(Object, ScheduleFor))
NEXT Object
```

Vergleich zu JOMP

Jede Erweiterung hat seine eigenen Vor und Nachteile

- Keine High-Level-Konstrukte wie Schleifenparallelisierung. Benutzer muss sich um das Verteilen auf mehrere Events selbst kümmern
- Paralle Abläufe entstehen durch das Anordnen von Events in parallelen Eventgruppen. JOMP verwendet nur parallele Regionen
- Gleichzeitige Verwendung von parallelen und sequenziellen Codeabschnitten, sodass keine Synchronisierungsprobleme innerhalb der Events entstehen können

Vergleich zu JOMP

Kleiner Codevergleich von Hydra und JOMP

- Javacode vor und nach der Schleife wird zuerst ausgeführt
- Hydra kann jede Schleifenimplementierung verwenden

Grund für diese relativ komplexen Konstruke: Vermeidung von Deadlocks!

Hydra version:

```
public @event void EventWithLoop() {
        EventGroup group = EventGroup.Parallel();
        group.AddAsChildEventGroup();
        group.AddEventsToThis();
        for (int a = 0; a < 10; a++)
                DoSomeWork(a);
}

public @event void DoSomeWork(@pass int a) {
        // Insert code for doing work here.
}
```

JOMP version:

```
public void MethodWithLoop() {
        //omp parallel for
        for (int a = 0; a < 10; a++)
                DoSomeWork(a);
}

public void DoSomeWork(int a) {
        // Insert code for doing work here.
}
```

Zusammenfassung

Übersicht der wichtigsten Sprachkonzepte:

Event group creation	Hydra command
FIFO	EventGroup.FIFO()
Parallel	EventGroup.Parallel()
Pre-scheduled	EventGroup.Prescheduled()
Sequential	EventGroup.Sequential()
Event group manipulation	**Hydra command**
Add event group *group* to the event group that events are currently being added to.	*group*.AddAsChildEventGroup()
Add event group *a* to event group *b*	*a*.AddAsChildEventGroupTo(*b*)
Add events to the event group *group*	*group*.AddEventsToThis()
Retrieve the event group that events are being added to.	EventGroup.GetAddingTo()
Retrieve the event group that the currently executing event belongs to.	EventGroup.GetCurrent()
Locking parameters	**Hydra command**
Pass	@pass
Read	@read
Write	@write
ReadWrite	@readwrite

Start eines Programms

- *.java Dateien werden in *.class Dateien umgewandelt. Normale Kompilierung
- Alchemist Compiler wird initialisiert und konfiguriert
- Alchemist Compiler liefert eine Kopie des Programms zurück
- Software kann mit java Hydra.Runtime.Start NameDesProgramms gestartet werden

Sehr einfach für den Benutzer bei dem Hydra erfolgreich auf dem Rechner eingerichtet ist

Eventgruppen Beispiel

Folgende Verschachtelung kann bei normalen Programmen oft auftreten:

```
ScheduleAllEvents():
        SCHEDULE SayHello()
        SCHEDULE SayGoodBye()
SayHello():
        DISPLAY "Hello"
        SCHEDULE SayHowAreYou()
SayHowAreYou():
        DISPLAY "How Are You?"
SayGoodBye():
        DISPLAY "Good Bye"
```

Bei Hydra werden Events zuerst gescheduled und später zum richtigen Zeitpunkt erst ausgeführt

Eventgruppen Beispiel - Fortsetzung

- **Ohne Hydra:** Ausgabe in folgender Reihenfolge - Hello, HowAreYou, GoodBye. Eventausführung entspricht dem Zeitpunkt des Scheduling
- **Sequenziell:** Hello, HowAreYou, GoodBye.
- **FIFO:** Hello, GoodBye, HowAreYou. Änlich wie bei sequenzieller Ausführung, jedoch werden geschedulte Events in einer Queue zur Ausführung hinten angehängt
- **Parallel:** Kann variieren und folgende Ergebnisse liefern: (1,2,3), (1,3,2) und (3,1,2)
- **Prescheduled:** Identisch zu parallel, da kein Gebrauch von forward oder backward gemacht worden ist

MergeSort Algorithmus

Gutes Beispiel für eine Prescheduled Eventgruppe

- Ohne forward und backward würden alle Events zur gleichen Zeit ausgeführt werden
- Bei MergeSort notwendig um korrekte Reihenfolge des "Mergens" zu erlangen

```
ScheduleMergeSort(Array):
    CREATE NEW Prescheduled Event Group
    ScheduleMergeSort(Array, 0, SizeOf(Array) - 1)

ScheduleMergeSort(Array, A, B):
    BACK
    IF A = B THEN
        RETURN
    Difference = B - A
    Middle = (A + B) / 2
    IF Difference >= G THEN
        ScheduleMergeSort(Array, A, Middle)
        ScheduleMergeSort(Array, Middle + 1, B)
        SCHEDULE Merge(Array, A, Middle, B)
    ELSE
        SCHEDULE MergeSort(Array, A, Middle, B)
    ENDIF
    FORWARD
```

Microbenchmark

Benchmarktests mit BAPS, welches bei Hydra direkt mitgeliefert wird um möglichst korrekte Benchmarks zu bekommen

- Verschiedene Implementierungen, da verschiedene Javaerweiterungen gebenchmarkt werden
- Als Testalgorithmus dient der Mergesort Algorithmus
- Je mehr paralle Hardware verfügbar ist, desto höher ist die Performance

	Desktop		Workstation		Server	
Test	Time (ms)	Faster	Time (ms)	Faster	Time (ms)	Faster
MergeSort with Java	152.04	× 1.00	155.66	× 1.00	865.20	× 1.00
MergeSort with JOMP	79.17	× 1.92	123.29	× 1.26	458.81	× 1.89
MergeSort with Hydra	92.42	× 1.65	72.82	× 2.14	305.06	× 2.84

Macrobenchmark

Auch hier wird mit BAPS gearbeitet

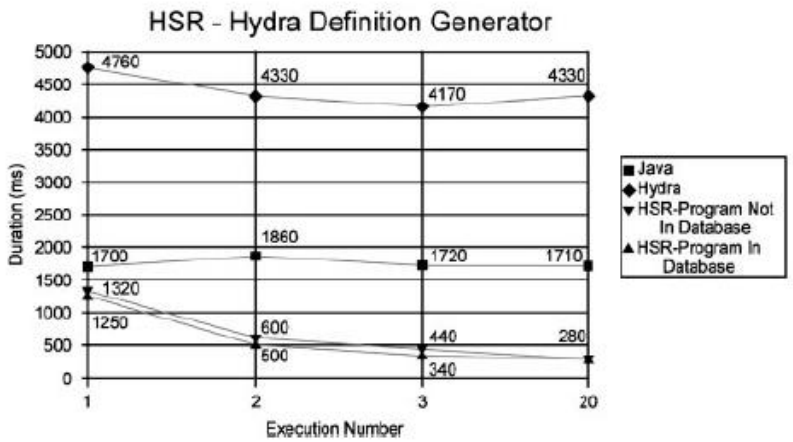
- Hydra Definition Generator ist ein Code Generator welcher einen Teil des Alchemistcompilers erstellt
- Kein JOMP Test vorhanden, da Code komplett anders implementiert werden müsste
- Performanceschub geringer, da viel I/O Elemente vorhanden sind und Reflection API ungeeignet füer Parallelität ist

	Desktop		Workstation		Server	
Test	Time (ms)	Faster	Time (ms)	Faster	Time (ms)	Faster
Definition generator with Java	247.27	× 1.00	325.36	× 1.00	1318.82	× 1.00
Definition generator with Hydra	211.96	× 1.17	233.35	× 1.39	877.34	× 1.50

Hydra Stay Resident (HSR)

Problem: Hydra Runtime benötigt lange zum initialisieren und Klassen müssen ebenfalls geladen und compiliert werden

- Bei HSR Einsatz bleibt Runtime nach der ersten Ausführung im Speicher
- Extremer Performanceschub unabhängig des Codes



Zusammenfassung

Hydra zeigt was möglich ist. Eine genaue Planung der parallelen Komponenten entfällt jedoch nicht.

- Laut Franklin E.Powers JR. ist Hydra nur Proof of Concept
- Gut geeignet für wissenschaftliche Forschungsprojekte
- Allerdings kaum private Verbreitung da komplexe individuelle Installation auf jedem Rechner vonnöten ist

Aber: Trotz mangelnder Alternativen eine der besten Erweiterungen für Java

Danke

Danke für ihre Aufmerksamkeit!