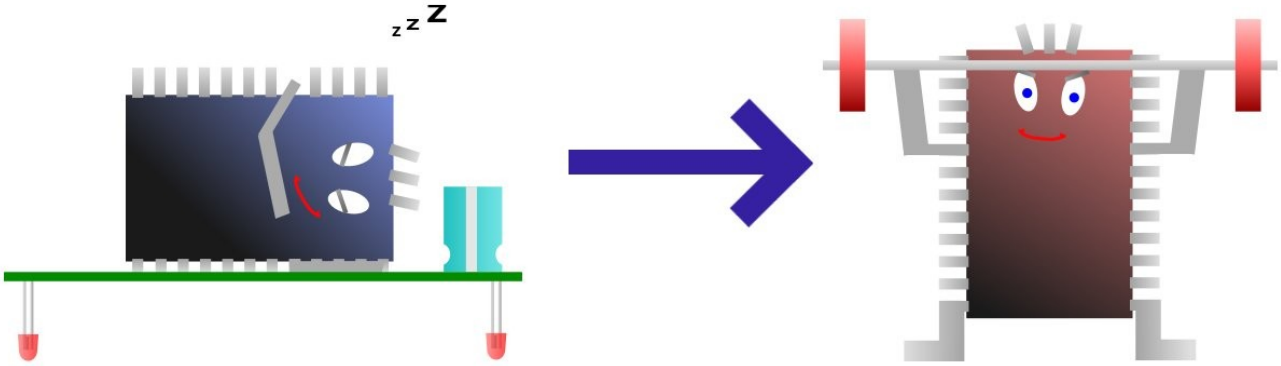
Parallellisering av Algoritmer för Flerkärniga Processorer



870928-3017 Johan Gustafsson
870303-4952 Gustaf David Hallberg
880525-8210 Per Hallgren
801117-0597 Wuilbert Lopez

Innehållsförteckning

Table of Contents

1 Bakgrund

Datortillverkning utmärks av två saker, prestanda och design. Kunden måste tilltalas av produkten, datorn, och samtidigt måste nya datorer prestera bättre än föregångarna. Vad gäller prestandaförbättring är minne (RAM) och processor centrala begrepp. I vårt projekt kommer vi inte att prata särskilt mycket om minne annat än om det är relevant för vår förståelse av en specifik algoritm.

Grunden till vårt projekt är att aktörer inom processorindustrin under det senaste decenniet har varit tvungna att sätta samman flera processorkärnor på samma chip istället för att producera snabbare processorer. Detta då höga klockfrekvenser (processorns arbetshastighet) innebär höga temperaturer, och att tempereraturerna i chippen blivit en begränsing. Man kan överkomma denna begränsning genom exotiska kylmetoder men kostnaden för kunden blir oacceptabel.

Idag återfinns processorer med flera kärnor inte bara i industrisammanhang utan i gemene mans persondator varför man idag bör se hur dessa resurser kan nyttjas på bästa sätt. Ett traditionellt program skrivs som en steg för steg instruktion, dvs seriellt. Program bör inte längre skrivas så att de exekveras seriellt. Istället bör man i sitt arbete som programmerare ta med i beräkningen att det finns mer än en kärna på den dator som ska köra det färdiga programmet och därmed skriva programmet med parallelliserad kod. Med parallelliserad kod kan delar av programmet köras på skilda processorkärnor oberoende av andra delar för att sedan låta resultat från de olika kärnorna sammanställas till det slutgiltiga resultatet. Parallelliserad kod kan därmed utföras snabbare än sekventiell kod (i en miljö med flera processorkärnor) med en faktor N, där N är lika med antalet processorkärnor.

2 Projektbeskrivning

2.1 Syfte

Vi ämnar att för en person utan datorvetenskapliga kunskaper visa hur parallellisering av sekventiella algoritmer på programspråknivå bättre kan nyttja dagens processorresurser. Fokus kommer ligga på en snabbare exekvering av terminerande algoritmer, det vill säga finita beräkningar. Det ideala målet är en linjär hastighetsökning som skalar med antalet kärnor. Kan man inte uppnå detta ska man ändå sträva efter detta mål i så stor utsträckning som möjligt. Rapporten avser även att redogöra för de svårigheter som föreligger vad gäller parallellisering samt generella metoder och riktlinjer för algoritmanalys.

2.2 Problem

Vi vill parallellisera en beräkningsintensiv algoritm som i regel körs över en – för användaren – märkbar tidsrymd. Det är viktigt att användaren upplever en märkbar prestandaökning varför vi finner det fördragbart att algoritmens arbetsprocess förmedlas visuellt. Exempelvis har vi haft bildredigeringsprogrammet GIMPs olika grafikfilter i åtanke; ”neonfilter”, ”ta-bort-röda-ögon-filter”, ”serietidningsfilter”, och så vidare.

Under uppgiftens gång kommer vi behöva analysera algoritmerna i fråga. Analysen kommer till stor del att gå ut på profilering. Profilering innebär att man mäter vilka delar av koden som används mest och vilka som tar mest tid. Detta mynnar sedan ut i en uppdelning av algoritmen där varje enskild del sedan var för sig kan analyseras vidare för att bestämma hur bäst man utför parallelliseringen.

Viktigt att notera är att programmeringens behovsträd fortfarande måste iaktas. Först och främst måste ett program ge korrekt resultat. Först därefter är kodens prestanda av intresse.

Uppgiften vi föresatt oss är i sin essens att ta befintlig (och korrekt) kod och se till att den delas upp på mer än en processor. Vi räknar med att vi kommer få korrekthetsproblem liknande de man får närhelst flera programprocesser är igång samtidigt på en enkärnig dator. Problem såsom låsning eller korrekt uppdelning av gemensamma minnesrymder.

2.3 Avgränsningar

Eftersom vi arbetar med algoritmerna på en programspråksnivå så kommer vi inte arbeta alltför hårdvarunära i hur algoritmerna kan köras snabbare, då skulle det kunna handla om att optimera minnescaching eller generella metoder för att utvidga godtyckliga algoritmer i hårdvara till flera kärnor så som det för några år sedan ryktades att företaget AMD arbetade med[1].

Vi kommer under projektets gång inte sätta oss in i teorin bakom de algoritmer vi jobbar mot, mer än så mycketUntitled spreadsheet som krävs för att göra en övergripande analys av deras förlopp och isolera kritiska delar som vi kan bryta ut.

1http://www.xbitlabs.com/news/cpu/display/20060622143710.html

3 Metod och genomförande

Vi har beslutat att exemplifiera hur parallelisering av ett program förbättrar dess prestanda genom att bearbeta simuleringsmiljön OpenFOAM som är ett projekt med öppen källkod skrivet i programspråket C++. OpenFOAM är ett mycket generellt verktyg för att utföra simuleringar. OpenFOAM ämnar att isolera programmering så mycket som möjligt för att underlätta för eventuella fysiker, kemister och mattematiker som skulle vilja utföra en simulering. Varje simuleringsproblem implementeras som ett fristående program, och det finns inga kopplingar mellan olika simuleringsproblem utan bara mellan varje problem och OpenFOAMs kärna.

Vi kommer att dela upp OpenFOAM i flera delar så att vi kan bearbeta kod relevant för olika simuleringsprogram var för sig och därmed arbeta effektivare. Vi delar dessutom upp arbetet i fyra steg enligt vår generella parallelliseringsmall:

1. Analys
    - Överskådning
    - Profilering
    - Punktanalys
    - Uppdelning
2. Parallellisering
    - Val av metod
    - Implementering
3. Verifiering
    - Säkerställande av resultatets överensstämmande med originalet
4. Testning
    - Prestandamätning
    - Profilering

Steg två till tre och två till fyra kan behöva upprepas om vi ej uppnått önskat resultat.

3.1 Analys

Analysfasen skapar en tydlig bild av hur programmet arbetar och var parallellisering kan göra mest nytta. Vi kommer att använda oss av gränssnittet OpenMP för att underlätta parallelliseringen, och vi använder profileringsverktyget GProf för att underlätta analys av hur programmet arbetar.

Till att börja med är det bra att överskåda projektet, dess mappstruktur och uppbyggnad. Detta kallar vi överskådning. När man fått en något klarare bild över hur projektet är upplaggt kan man gå vidare till profilering.

3.2 Parallellisering

Efter att analyserat och satt sig in i projektet, är det dags att parallellisera kod. I detta steg kommer vi försöka parallellisera både kärnan och simuleringsprogrammet i sig i den mån det går. Vi gör detta för att parallellisering introducerar ett visst extraarbete för processorn och i praktiken kan det innebära att ett stycke kod i kärnan som anropas ofta men som vid varje anrop körs över små tidsrymder inte tjänar på att delas upp över flera kärnor, tvärtom så kan man förlora prestanda. Om man istället delar upp alla dessa anrop i simuleringsprogrammet i sig kommer inte varje anrop delas upp, utan alla anrop i sin helhet sprids istället ut på de tillgängliga kärnorna. Vid detta förfarande

riskerar man istället att betala en större minneskostnad. Detta är alltså två olika metoder, och vi kommer att implementera båda för att kunna testa dem och se vilken som är effektivast.

3.3 Verifiering

För att parallelliseringen skall vara korrekt räcker det inte med att den är snabbare. Det parallelliserade programmet måste också ge samma resultat som originalet. Vi vill säkerställa ett exakt överensstämmande och vara säkra på att vi inte luras av snarlikt resultat eller tillfälliga överensstämmanden.

3.4 Testning

Eftersom det här projektet i sin helhet går ut på att effektivisera ett program, är den här delen väldigt viktig. Vi kommer att testköra våra program noga. Till att börja med på våra privata datorer med upp till 4 kärnor, och som grand finale på en 64-trådad dator från Sun.

4 Tidsplan

- 25/3 Torsdag — Möte med handledare. Besluta om vi ska parallellisera kärna eller modul.
- Påsklovet — Viss mån av parallellisering uppnådd
- 12/4 Måndag — Möte, uppslutning.
- 12/4 — 18/4 — Testa på Sun-maskinen.
- 19/4 Måndag — Möte
- 26/4 Måndag — Möte
- 3/5 Måndag — Möte, Börja arbeta med rapporten
- 10/5 Måndag — Möte
- 17/5 Måndag — Möte
- 18/5 Tisdag — Inlämning av rapport
- 24/5 Måndag — Möte
- 26/5 Onsdag — Inlämning av opposition (skriftlig)
- 31/5 Måndag — Muntlig slutredovising
- 1/6 Tisdag — Muntlig slutredovising