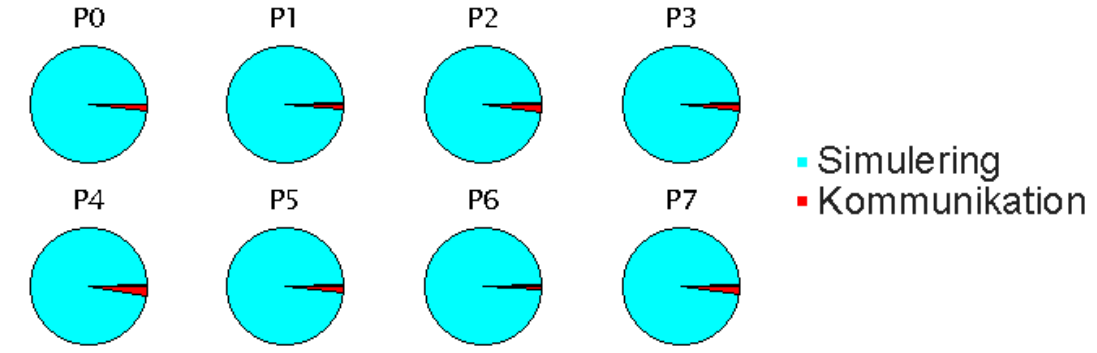
Christoffer Nylén, chrny719@student.liu.se
Alexander Norström, aleno278@student.liu.se

Tools – TotalView

Vi använde oss av TotalView i tråd labben och partikelsimuleringen för att lösa segmenteringsfel i programmen. Som i andra debug verktyg gick det att sätta brytpunkter och inspektera lokala tillstånd under körning. TotalView tillät en inspektera programmet på olika trådar samt processorer. Vilket var praktiskt vid parallellprogrammeringen, då vi kunde studera tillstånden i de olika processerna för att kunna felsöka kommunikations- och logiska fel. Det grafiska gränsnittet var bökigt att använda och det var svårt att hitta det man letade efter. Gdb är snällare på den frontent när man vill inspektera minne. I TotalView så fick vi gissa pekarförhållanden ett par gången för att inspektera element i pekare till minnesutrymmen med flera element, eftersom den endast visade första elementet och inte gav alla element i listan. Detta är enklare i gdb där man direkt kan ange adress eller pekare samt längden på antal element som ska hämtas. Vi tycker att Visual Studios debugger är trevligare att arbeta med och Visual Studio kommer även med en profiler för att studera hur programmet exekveras utan att behöva skriva extra kod för att kunna studera dessa saker som vi gjorde med VT för ITAC. Det känns rimligt att man ska kunna studera detta även på parallella program, åtminstone för program på SMP processorer.

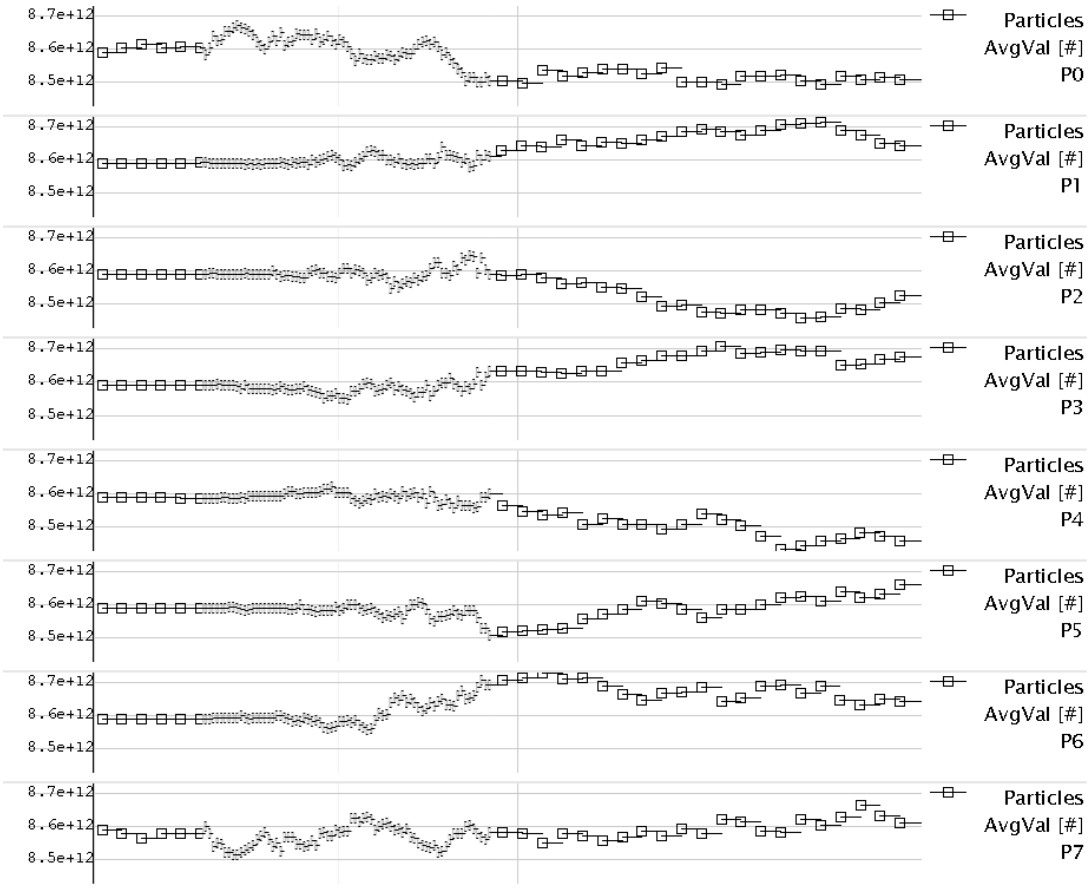
Tools - ITAC

Vi ser att last fördelningen blev jämn. Röd indikerar tid spenderad på kommunikation och turkos visar tid spenderad i partikel simulering.



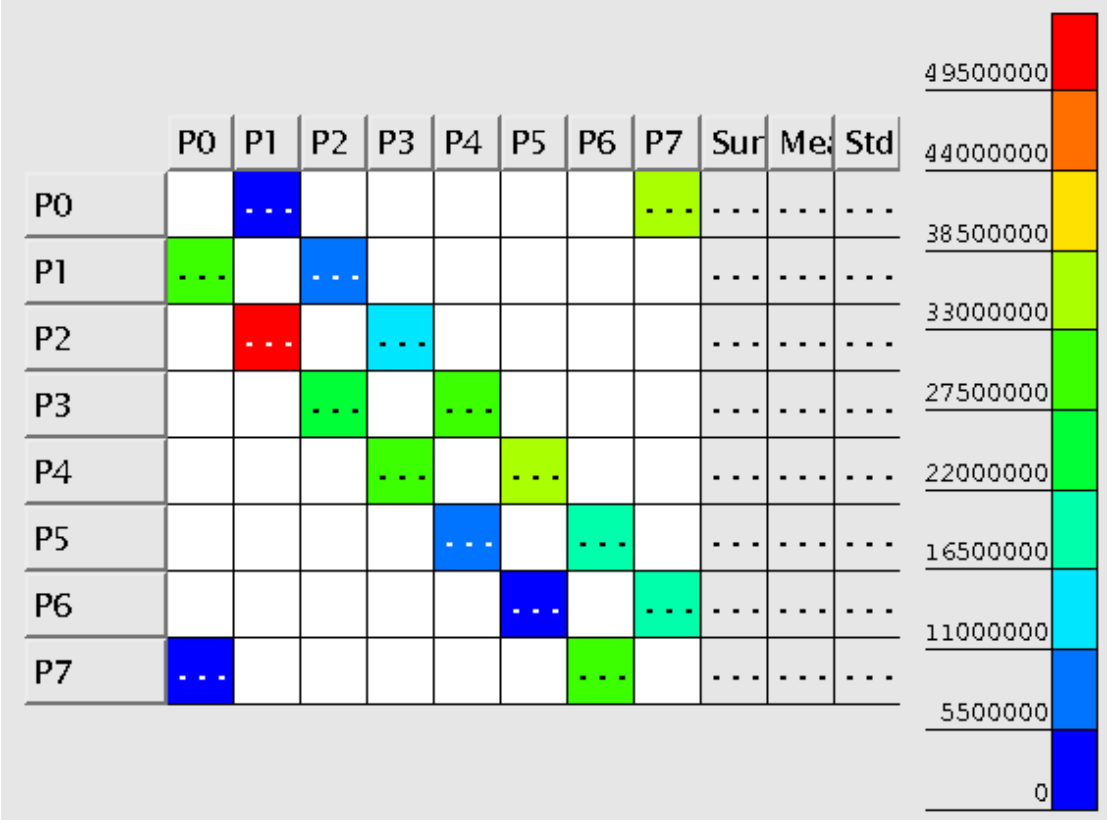
Figur 1: Load balancing

Vi ser att antalet partiklar per kärna håller sig på en jämn fördelning (mellan 8.73e+12 och 8.44e+12 vilket motsvarar 2030 respektive 1965 partiklar per kärna) när vi kör totalt 16000 partiklar.



Figur 2: Antal partiklar per kärna

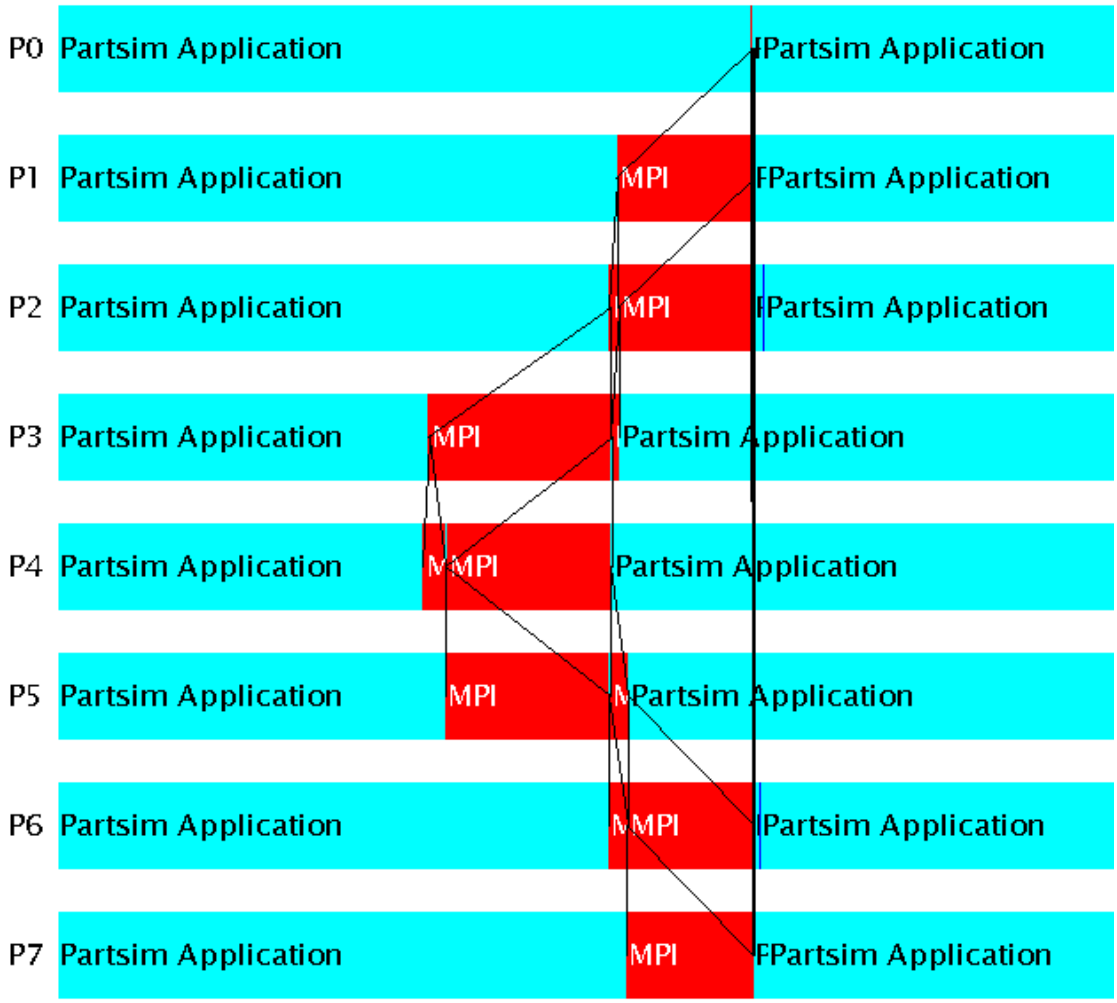
Christoffer Nylén, chrny719@student.liu.se
Alexander Norström, aleno278@student.liu.se



Figur 3: Kommunikation mellan processer

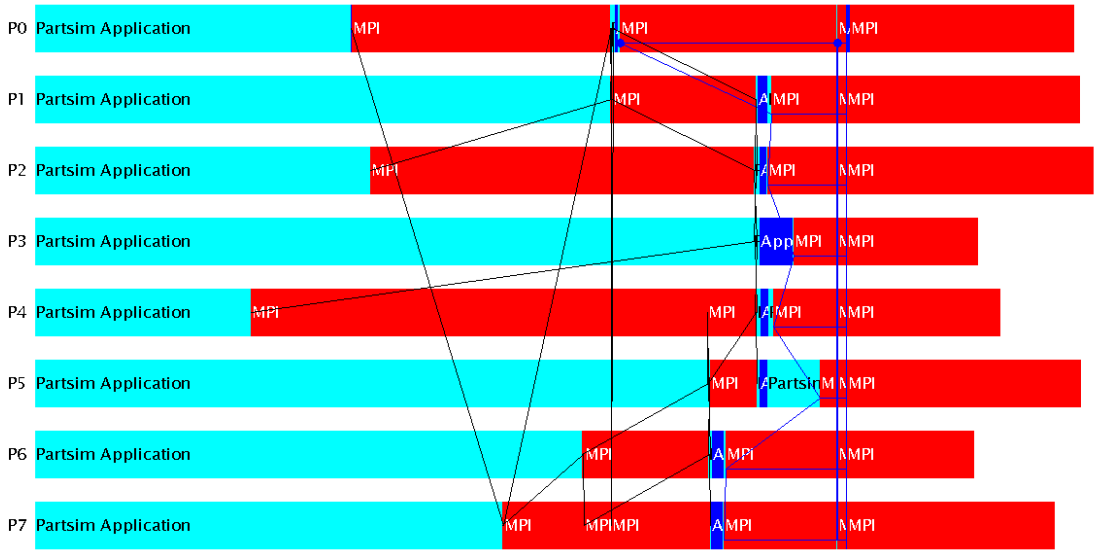
I figuren för kommunikation mellan processerna så ser vi att kommunikationen förekommer på diagonalen. Detta beror på att vi utbyter kommunikationen enligt en ring topologi där vi pratar med den närmsta grannen. I diagrammet kan vi se hur mycket partiklar som skickas från en process till en annan. Vi ser att vissa processer inte skickar så mycket vilket betyder att de är få partiklar som lämnar domänen hos dessa processer. Vi har även en väldigt högt lastad process P2 till P1 som kan tolkas som att det är väldigt många partiklar i P2 som flyr väster ut i P2's domän.

Christoffer Nylén, chrny719@student.liu.se
Alexander Norström, aleno278@student.liu.se



Figur 3: Dubbellänkad kommunikation

I Figur 3 så visas hur processerna kommunicerar med varandra under varje iteration för att utbyta information om hur partiklar förflyttar sig. Detta skrer enligt ett ringmönster vilket syns i bilder som att processorerna först skickar till vänster och tar emot från vänster och därefter skickar till höger och tar emot till höger. Den information som sedan ska växlas mellan processerna skickas sedan direkt efter. Eftersom att Recieve är blockernade så ger det bieffekten att noderna synkroniseras vid partikelutbytet, innan de kör vidare.



Figur 4: Reducering vid beräkning av momentet

I slutet av programmet sker en reducering för att beräkna det totala momentet. Figur 3 och 4 illustrerar hur kommunikationen sker, där sträcken visar hur meddelanden skickas mellan olika processer.