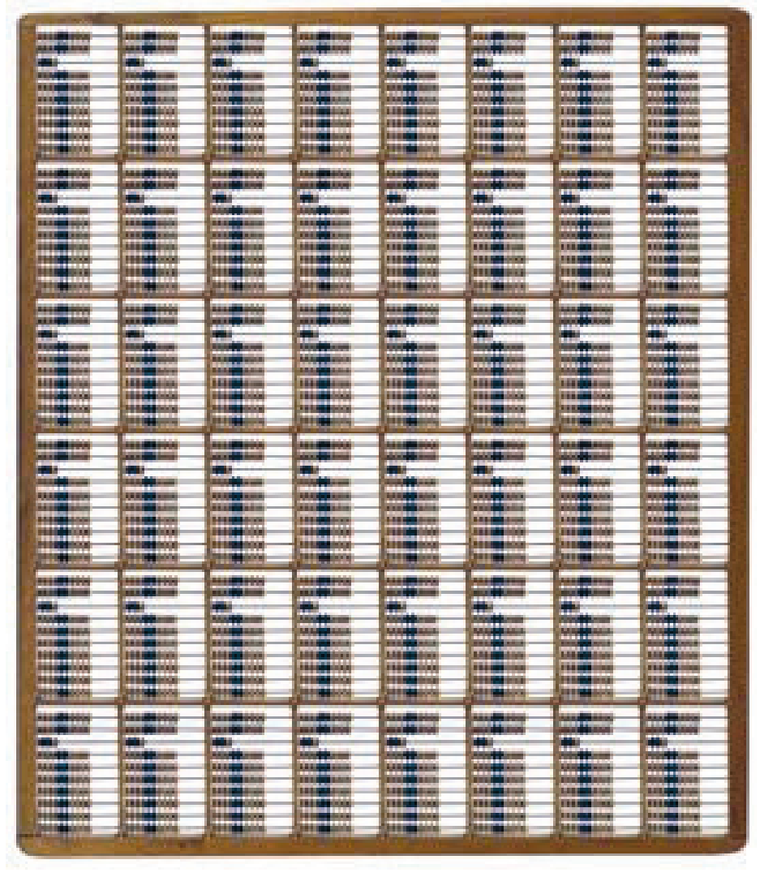
Användning av grafikkort för lösenordstestning

JACOB LÖFVENBERG

EFFEKTIVA BERÄKNINGAR GENOM MASSIV PARALLELLISM



FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1000 anställda varav ungefär 800 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömning av olika typer av hot, system för ledning och hantering av kriser, skydd mot och hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Informationssystem
Box 1165
581 11 Linköping

Tel: 013-37 80 00
Fax: 013-37 81 00

www.foi.se

FOI-R--2966--SE
ISSN 1650-1942

Metodrapport
December 2009

Informationssystem

Jacob Löfvenberg

Användning av grafikkort för lösenordstestning

Effektiva beräkningar genom massiv parallellism

Titel	Användning av grafikkort för lösenordstestning
Title	Using graphic cards for password testing
Rapportnr/Report no	FOI-R--2966--SE
Rapporttyp Report Type	Metodrapport/Methodology report
Månad/Month	December
Utgivningsår/Year	2009
Antal sidor/Pages	29 p
ISSN	ISSN 1650-1942
Kund/Customer	Försvarsmakten
Projektnr/Project no	E71344
Godkänd av/Approved by	

FOI, Totalförsvarets Forskningsinstitut
Avdelningen för Informationssystem
Box 1165
581 11 Linköping

FOI, Swedish Defence Research Agency
Information Systems
P.O. 1165
SE – 581 11 Linköping

Sammanfattning

WLAN är ett allt vanligare sätt att bygga lokala nät. Kommunikationen över sådana nät skyddas i allmänhet med WPA2-kryptering som bygger på underliggande starka kryptoalgoritmer som anses omöjliga att forcera. Det som går att göra är att angripa modellen för nyckeldistribution eftersom denna ofta bygger på lösenord som användarna och nätägarna måste mata in i alla enheter som ska delta i kommunikationen. Genom att först avlyssna när en nätenhet kopplar upp sig mot nätet går det att gissa lösenord och avgöra när rätt lösenord har hittats. Detta kan göras utan att kommunicera med nätet, vilket öppnar möjligheten för angriparen att använda godtyckligt stor beräkningskraft i sökandet efter rätt lösenord. WPA2 har ett visst inbyggt skydd mot sådana gissningsattacker, vilket gör att det är en beräkningskrävande uppgift att testa många lösenord, varför lösningar som är snabbare än standarddatorer är önskvärda.

Ett sätt att snabba upp beräkningar, som utvecklats snabbt de senaste åren, är att använda den stora parallelliteten i moderna grafikkort. De snabbaste grafikkorten har hundratals beräkningsenheter som arbetar samtidigt. Dessa är visserligen begränsade i ett antal avseenden, men för vissa typer av problem så är detta inget stort hinder. Grafikprocessorerna blir dessutom allt mer flexibla för varje ny generation, vilket gör att allt fler problem går att lösa effektivt i grafikhårdvara.

I rapporten beskrivs vilken beräkningsprestanda som kan uppnås, både med traditionella CPU:er och med de modernaste grafikkorten. Detta görs i viss mån teoretiskt, men framför allt beskrivs hur en specialbyggd grafikkortsberäkningsmaskin som byggts vid FOI presterar vid lösenordsgissning i WPA2. Som förväntat är den markant snabbare än lösningar som bygger på vanliga CPU:er, även om skillnaden inte är så stor som de teoretiska analyserna antyder är möjligt.

Avslutningsvis beskrivs i rapporten hur FOI satt samman en lista med lösenordsförslag. Det visar sig att trots att listan hämtar innehåll från ett antal ganska omfattande källor så blir slutresultatet en lista av hanterbar storlek – drygt 1,5 miljoner element. Det är inte förrän listan expanderas med olika typer av

manipulationer och förändringar av elementen som antalet lösenordsförslag blir riktigt stort. Ett kluster med 40 maskiner lika den som FOI byggt skulle dock kraftfullt nog att inom rimlig tid arbeta sig igenom åtminstone de enklaste expanderade listorna. En öppen fråga som lämnas för framtida forskning är hur bra listor av den här typen är, i betydelsen hur stor andel av faktiskt använda lösenord som finns med i listorna.

Nyckelord: lösenord, GPGPU, grafikkort

Summary

WLAN is an ever more common way of building local networks. Communication in such networks are usually protected by WPA2 encryption, which is built upon strong encryption algorithms that are usually seen as impossible to crack. What is possible to attack is the key distribution model since this usually builds upon the use of passwords, which the network owner and the users have to enter into every node that will participate in the communication.

By first eavesdropping when a network node connects to the network it is possible to test passwords and decide when the correct password has been found. This can be done without communicating with the network, which enables the attacker to use an arbitrary amount of computing power in the search for the correct password. WPA2 has a certain level of protection against such guessing attacks, which makes it an arduous task to test many passwords. This makes the use of solutions faster than standard computers attractive.

One way of speeding up computations, that has progressed quickly in the last few years, is using the parallelism in modern graphics cards. The faster graphics cards have hundreds of computation units working simultaneously. These are limited in a number of ways, but for some types of problems this is no great hindrance. Graphics processing units are also growing more flexible with each new generation, which means that ever more problems can be efficiently solved in graphics hardware.

The report describes what performance that can be achieved, both using traditional CPUs and using the latest generation of graphics cards. This is partly done theoretically, but more importantly, it is described how an FOI built computer for graphics cards computations performs in password guessing in WPA2. As can be expected it is significantly faster than solutions using standard CPUs, even though the difference is not as great as the theoretical analysis would suggest.

Finally the report describes how FOI has compiled a list of possible passwords. What is found is that even though the list is synthesized from several rather large

sources, the end result is a list of managable size – some 1.5 million elements. It is not until the list is expanded using different types of manipulations and changes of the list elements that the number of possible passwords grows really large. A cluster with 40 computers like that built by FOI is however powerful enough to process at least the simplest expanded lists within a reasonable time. An open question which we have left for future work is how good lists like these are, in the sense of how large a part of actually used passwords is in the lists.

Keywords: passwords, GPGPU, graphics cards

Akronymer och förkortningar

A5/x – En serie kryptoalgoritmer som används i GSM-systemen. Idag finns A5/1, A5/2 och A5/3.

AES – *Advanced Encryption Standard*, en civil kryptoalgoritm som standardiserats av USA.

AMD – en amerikansk, multinationell halvledartillverkare som gör bl.a. CPU:er och grafikkort.

ASCII – *American Standard Code for Information Exchange*, ett vanligt sätt att representera tecken

C – ett programspråk, ursprungligen skapat 1972 för att använda till operativsystemet UNIX.

CAD – *Computer Aided Design*, mjukvarubaserat datorstöd för design .

CPU – *Central Processing Unit*, den komponent i en dator som utför instruktioner och beräkningar.

CUDA – *Compute Unified Device Architecture*, parallelldatorarkitektur utvecklad av Nvidia.

DES – *Data Encryption Standard*, en kryptoalgoritm som standardiserats av USA. Ersatt av AES.

FFT – *Fast Fourier Transform*, en effektiv algoritm för att beräkna den diskreta fouriertransformen.

FOI – Totalförsvarets Forskningsinstitut.

GFLOP – *Gigaflop*, enhet för beräkningsvolym; miljarder flyttalsoperationer.

GNU – *GNU's not Unix*, GNU-projektet är ett ”paraplyprojekt” för utveckling av fri programvara.

GPL – *General Public License*, GNU GPL är en mjukvarulicens för fri programvara.

GPU – *Graphics Processing Unit*, en processor specialicerad för grafikberäkningar.

GSM – *Global System for Mobile communications*, den mest spridda standarden för mobiltelefoni.

HD – *High Definition*, förhöjd visuell upplösning. Full-HD är 1920x1080 pixlar.

OpenCL – *Open Computing Language*, ett ramverk för programutveckling för heterogena miljöer.

PCI-express **–** *Peripheral Component Interconnect Express*, gränssnitt för expansionskort i datorer.

PSK – *Pre-Shared Key*, en metod i WPA för att skapa och distribuera kryptonycklar.

RADIUS – *Remote Authentication Dial In User Service*, nätprotokoll för autentisering med mera.

RC4 – en mycket vanlig algoritm för strömchiffer som lämpar sig väl för mjukvaruimplementation.

SHA1 – *Secure Hash Algorithm 1*, en kryptografisk hashalgoritm som standardiserats av USA.

SIMD – *Single Instruction, Multiple Data*, en teknik för dataparallellism i beräkningar.

SSE2 – *Streaming SIMD Extensions 2*, en uppsättning SIMD-instruktioner som finns i Intels CPU:er.

SSID – *Service Set Identifier*, ett namn som identifierar ett visst WLAN.

Wi-Fi – varumärket för en viss organisation som arbetar för att sprida användningen av WLAN.

WLAN – Wireless Local Area Network, ett trådlöst, lokalt datanät.

WPA – *Wi-Fi Protected Access*, en certifiering som visar att ett WLAN uppfyller vissa säkerhets-krav. Används också som beteckning på själva säkerhetsmetoderna. Idag ersatt av det nyare WPA2.

Innehållsförteckning

1 Bakgrund

FOI har i detta arbete som finansierats av Försvarsmakten studerat möjligheten för en angripare att avlyssna WLAN-kommunikation. En del i att avlyssna sådan kommunikation är att angriparen måste ta sig förbi det kryptografiska skydd som i allmänhet används på sådana förbindelser, vilket är vad vi diskuterar i denna rapport. Den äldre skyddsmetoden WEP (Wired Equivalent Privacy) är idag att betrakta som knäckt i alla avseenden varför vi inte behandlar den i rapporten. Istället vänder vi oss mot de nyare metoderna WPA och WPA2 (Wi-Fi Protected Access), där den senare är den som idag är helt dominerande.

2 WPA/WPA2

2.1 Övergripande funktion

WPA och WPA2 är två versioner av samma protokoll och används för att skydda WLAN-kommunikation mot avlyssning. WPA/WPA2 var ett svar på de brister som började upptäckas i WEP-protokollet i början av 2000-talet och som i dag lett till att WEP kan forceras i mjukvara på bara några minuter. WPA/WPA2 är såvitt är känt ett betydligt bättre protokoll, även om det under 2008 publicerades en kryptoanalytisk attack som kan användas mot korta paket med huvudsakligen känt innehåll, till exempel ARP-paket. Det finns dock ingen känd attack mot nyttokommunikation i WPA/WPA2.

WPA2 presenterades 2004 och är idag standardskyddet för WLAN-kommunikation. Kryptot i WPA2 baseras på AES och bör därmed vara att betrakta som säkert. Kryptoanalytiska svagheter får antagligen sökas på protokollnivå istället.

I WPA/WPA2 finns två sätt att distribuera kryptonycklar till de enheter som ska kommunicera i samma nät. I den första metoden används en RADIUS-server för att distribuera nycklar till de enheter som behöver. Denna metod är anpassad för stora organisationer och kräver att alla enheter redan har någon sorts certifikat eller engångslösenord. Den andra metoden kallas Pre-Shared Key (PSK) och är tänkt för hemmanät och mindre organisationer. PSK-metoden bygger på att en lösenfras används för att skapa en kryptonyckel. Alla måste alltså känna till samma lösenfras. Oavsett vilken metod som används resulterar den i en 256-bits kryptonyckel, vilket innebär att den är för stor att angripa om den är genererad på ett bra sätt.

Den attack som i praktiken har studerats och använts mot WPA/WPA2 är lösengissning, vilket bara fungerar mot PSK-metoden för nyckeldistribution. Om ett ”lätt” lösen använts är det förstås möjligt att gissa det. Vi kommer i fortsättningen bara att diskutera WPA/WPA2-PSK och lämnar den mer avancerade RADIUS-lösningen.

2.2 WPA/WPA2-PSK

2.2.1 Säkerhetsmodell

I WPA/WPA2-PSK anges en lösenfras som omvandlas till en 256-bits kryptonyckel som, efter ett ganska avancerat handskakningsförfarande, resulterar i den 256-bits kryptonyckel som faktiskt används för kommunikationen. Standarden säger att lösenfrasen måste vara minst 8 och högst 63 ASCII-tecken

lång. I praktiken kan varje tecken väljas bland ungefär 100 symboler, det vill säga motsvarande knappt sju bitars information. Alltså krävs minst 36 tecken i lösenfrasen för att generera en nyckel med 256 bitars entropi (slumpinformation). I praktiken kommer de flesta användare att använda betydligt kortare lösenfraser eftersom det är svårt att komma ihåg och jobbigt att distribuera och mata in långa lösenfraser. Entropin i 36 tecken räcker dessutom bara till om alla dessa väljs slumpmässigt med lika sannolikhet för alla alternativ, något som normalt inte alls gäller för lösenfraser. Även en idealt vald lösenfras innehåller, om den endast är åtta tecken lång, bara 56 bitars information, det vill säga ungefär en femtedel av kryptonyckelns storlek. Detta innebär visserligen inte att den resulterande 256-bitarsnyckeln innehåller 200 nollor, utan följden blir att kryptonyckeln under omvandlingen från lösenfrasen kommer att väljas bland bara 2^{56} nycklar. Tillsammans gör detta att det är möjligt att attackera WPA/WPA2-PSK genom att gissa lösenfraser.

2.2.2 Attacker

Om en motståndare avlyssnar handskakningsförfarandet mellan två enheter som ska kommunicera med WPA/WPA2-skydd så framgår inget om den nyckel som i slutändan kommer att användas. Däremot så går det utifrån denna avlyssnade information att kontrollera om ett föreslaget lösen är korrekt, utan att behöva kommunicera med någon av de avlyssnade enheterna. Det är alltså möjligt att testa lösenfraser off-line, utan inblandning av någon annan än angriparen. Detta är mycket fördelaktigt för denne, eftersom det enda som begränsar en sådan attack är beräkningskraften hos angriparen, vilket i sin tur bara beror på budgeten. Angriparen kan alltså bearbeta problemet helt på sina egna villkor.

2.2.3 Skydd

De som designat WPA/WPA2 har varit medvetna om de möjligheter som angriparen har och byggt in ett visst skydd i systemet. Det viktigaste elementet i detta skydd är att algoritmen som används för att bygga kryptonyckeln från lösenfrasen är relativt beräkningstung. Detta uppnås i WPA/WPA2-PSK genom en algoritm som heter PBKDF2 och som i princip innebär att lösenfrasen tillsammans med SSID (nätets namn) SHA1-hashas 4096 gånger och det slutliga resultatet används till kryptonyckeln (egentligen som ingångsvärde i handskakningsprotokollet). 4096 SHA1-hashningar låter mycket men i moderna system så tar detta bara en bråkdel av en sekund och det behöver bara göras när lösenfrasen byts ut. För en angripare innebär det dock att varje lösenfras som ska testas resulterar i 4096 hashningar, något som gör att testningen går väldigt mycket långsammare än den annars gjort, vilket i sin tur gör att det i praktiken blir omöjligt att testa alla möjliga lösen ens om det är valt med minimala åtta tecken (se avsnitt 5.2, Räkneexempel för test av lösenlistor).

Den andra delen i skyddet mot gissande angripare är att namnet på nätet, SSID, används som ”salt” och länkas samman med lösenfrasen innan kryptonyckeln beräknas. Detta medför att en angripare inte kan utgå från en lösenfras, beräkna kryptonyckeln och testa den mot flera avlyssnade nät samtidigt. Eventuella förberäknade databaser för lösenfraser blir också specifika för varje SSID vilket gör att sådana databaser kräver större lagringsutrymme och större beräkningsarbete.

Prestandaeffekterna som resulterar av denna balans mellan attackmöjligheter och skydd framgår i avsnitt 5.2, Räkneexempel för test av lösenlistor.

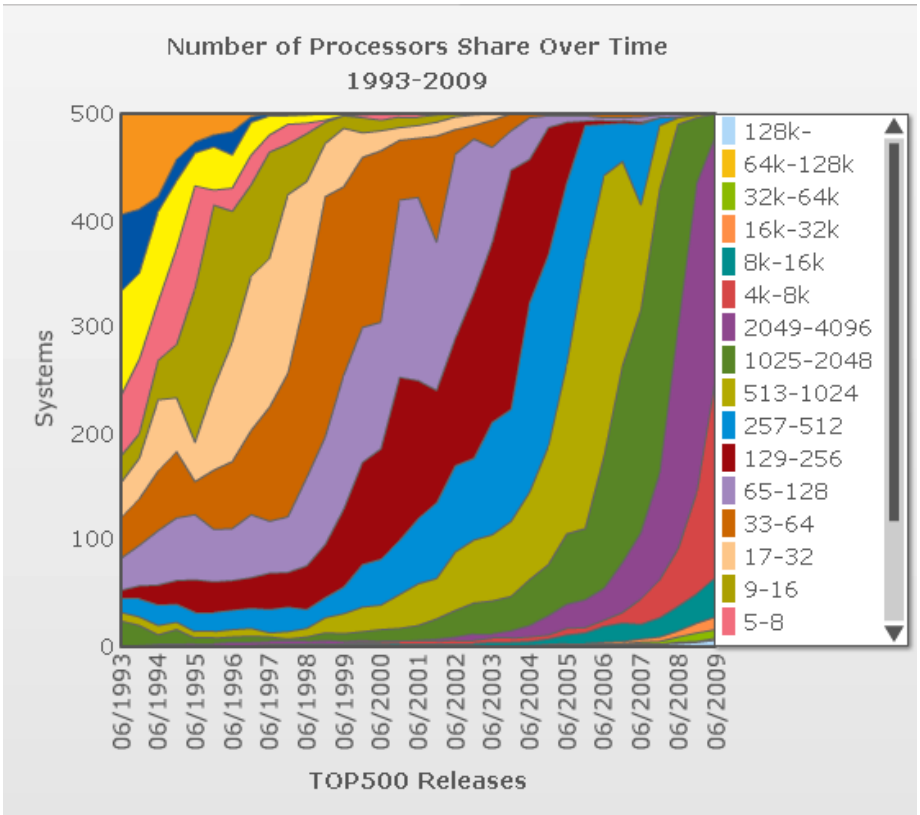
3 Grafikkortsberäkningar

I det föregående kapitlet framgår att det är mycket beräkningskrävande att testa lösenfraser för att forcera WPA/WPA2-PSK. Det finns olika sätt att snabba upp ett sådant förfarande. Ett sätt som blivit allt mer använt de senaste åren för att snabba upp många sorters beräkningar är att använda moderna grafikkort eftersom dessa idag är mycket snabba. I detta kapitel undersöker vi vilka möjligheter en sådan lösning erbjuder och hur man går tillväga för att använda den.

3.1 Inledning

Beräkningsprestanda i datorer har under flera decennier genomgått en stadig och extremt snabb förbättring. En fördubbling var 18:e till 24:e månad har varit normen. Detta har lett till att moderna CPU:er är mycket snabba och att i princip vilken CPU som helst duger till nästan alla uppgifter en normal användare utnyttjar datorer till. I vissa sammanhang finns dock fortfarande ett behov av datorer som är så snabba som möjligt eftersom problemen är beräkningsbegränsade.

En trend som blivit allt tydligare de senaste tio åren är att antalet beräkningsenheter i de snabbaste datorerna har ökat. Ett exempel går att hitta i den historiska statistiken över de 500 snabbaste datorerna i världen (http://top500.org). Där framgår att 2004 hade ungefär 95 % av de 500 snabbaste datorerna färre än 2000 CPU:er. Idag är det ungefär 95 % som har *fler* än 2000 CPU:er. I den något stökiga figuren nedan, tagen från samma webbplats, framgår hur påtaglig trenden är.

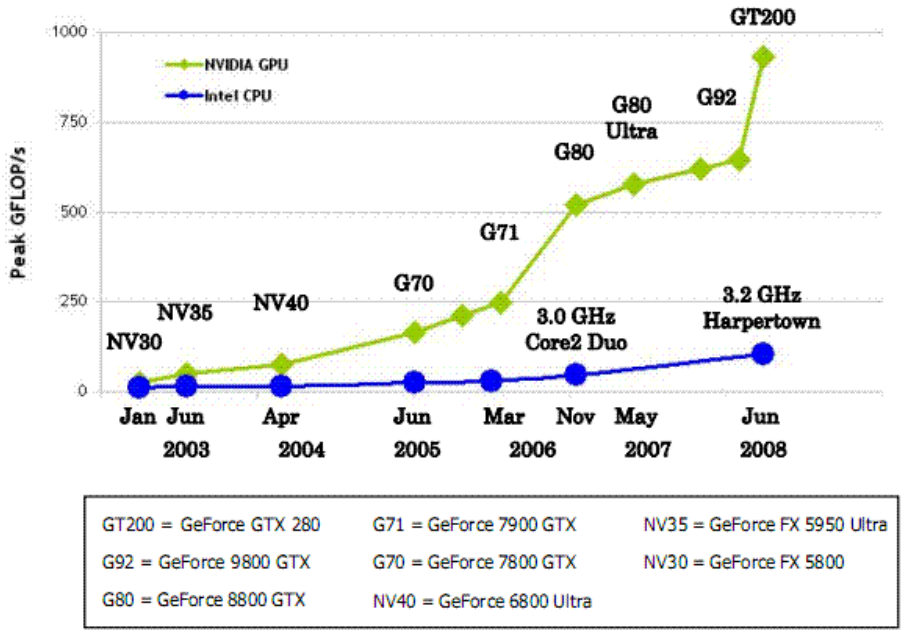


Figur 1: Antal av världens 500 snabbaste datorer med olika antal processorer. Källa: http://top500.org.

Trenden med fler beräkningsenheter har även nått persondatorer och servermaskiner i form av CPU:er med flera kärnor – i dagsläget mellan två och fyra och i närtid upp till åtta kärnor.

Parallellt med den snabba utvecklingen hos CPU:erna har grafikkortens beräkningsenheter, GPU:erna (graphics processing unit), utvecklats ännu fortare de senaste 10–15 åren. Det finns i grafiksammanhang ett inneboende behov av enorm beräkningskraft. Tag till exempel en full-HD-bild med cirka 2 Mpixel. Med en uppdateringsfrekvens på 50 Hz så är det 100 Mpixel per sekund som ska presenteras. I en avancerad grafisk miljö kan varje pixel behöva 100–1000 instruktioner för att beräknas, till vilket ska läggas de beräkningar som behövs för att manipulera 3D-världen innan den övergår till pixlar. En storleksordning på 100 miljarder instruktioner per sekund är alltså inte orimligt, vilket förklarar varför efterfrågan på snabbare grafikkort inte mattats trots den extremt snabba

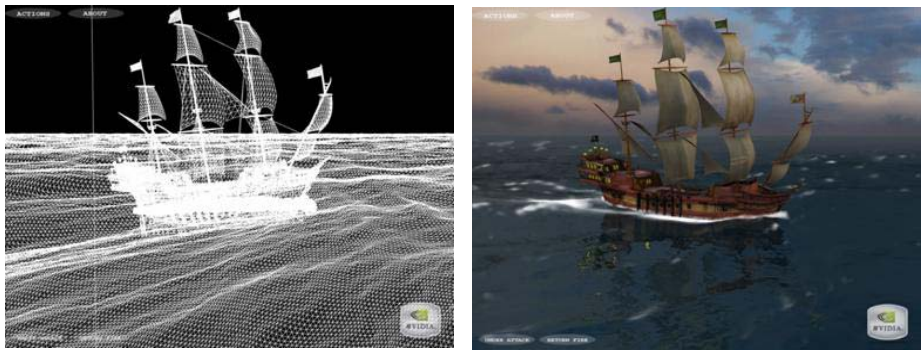
utvecklingen på området; en utveckling som varit markant snabbare än på CPU-sidan. I figuren nedan visas en jämförelse mellan Intels CPU:er och Nvidias GPU:er fram till mitten av 2008.



Figur 2: Teoretisk maxprestanda hos Intels CPU:er och Nvidias GPU:er fram till juni 2008. Källa: Nvidia.

3.2 Grafikhårdvara

Ett grafikkort omvandlar en tredimensionell beskrivning av en scen till pixelinformation för den bildskärm resultatet ska presenteras på. I figuren nedan syns ett exempel på hur en modell renderats till en färdig bild. Exemplet kommer från Nvidia.



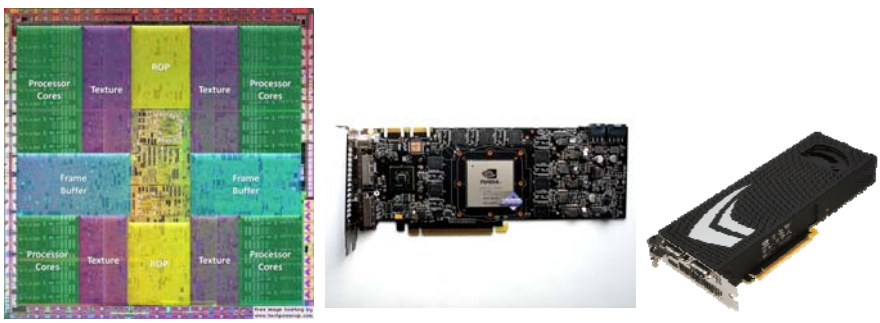
Figur 3*:* Modell och färdig rendering av ett segelfartyg. Källa: Nvidia.

Omvandlingen från 3D-modell till pixelbild sker i flera steg. Grovt kan följande steg urskiljas:

- vytransformation – där 3D-modellen flyttas in i ett nytt koordinatsystem som baseras på den tänkta kamerans placering och riktning. Detta ger en 3D-beskrivning som är korrekt ut betraktarens synvinkel.
- projektion – där 3D-modellen projiceras ner på en tänkt tvådimensionell yta framför den tänkta kameran. Denna yta kan ses som en abstraktion av bildskärmens yta.
- rastrering – där objekten på 2D-ytan omvandlas till fragment (”förpixlar” som kommer att behandlas vidare) i ett 2D-raster som är direkt kopplat till skärmens pixel-yta.
- pixel-shading – där det slutliga värdet för varje pixel beräknas utgående från färg, djup, position, belysning och textur (ytmönster). Denna beräkning kan vara komplex och berör varje individuell pixel, vilket har lett till att denna enhet är mycket kraftfull på moderna grafikkort.

Det var inte förrän 1999 som hela grafikrenderingsprocessen implementerades i konsumenthårdvara första gången. Detta gjordes i Nvidias grafikkort GeForce 256 som med sina 23 miljoner transistorer och 120 MHz var snabbare än allt som setts tidigare. Redan då började de första försöken med att få GPU:n att göra andra beräkningar än sådana som var grafikrelaterade, men det var ett fåtal individer som gjorde det och det betraktades snarast som att ”hacka GPU:n”. Under tvåtusentalet har generella beräkningar i grafikhårdvara varit ett växande intresse som ridit på den snabba utvecklingen inom området. Från cirka 2005, när Shader Model 3.0 introducerades i hårdvaran i GeForce 6-serien (cirka 200 miljoner transistorer och 400 MHz) så brukar man prata om GPGPU, General Purpose Computing on Graphics Processing Units. Den nya shaderförmågan möjliggjorde någorlunda generella beräkningar i de upp till 16 shaderenheterna. I

november 2009 kommer nästa generation grafikkort från Nvidia, GeForce 300-serien. Förhandsinformationen anger att dessas GPU:er kommer att ha 3 miljarder transistorer och 512 shaderenheter som ska köras i cirka 1,5 GHz. Den beräknade maxprestandan är cirka 2500 GFLOP per sekund (vid ”rätt” beräkningar kan flera instruktioner utföras samtidigt i hårdvaran). Motsvarande värden för den nuvarande snabbaste serien grafikkort från Nvidia, GeForce 200, är 1,4 miljarder transistorer och 240 shaderenheter som körs i cirka 1,4 Ghz, resulterande i en maxprestanda på ungefär 1000 GFLOP per sekund. Detta kan jämföras med att en av de snabbaste versionerna av Intels senaste processorgeneration, Intel i7, presterar lite drygt 50 GFLOP per sekund. I figuren nedan visas hur chippet som sitter i GeForce 200-serien (GT200b-chippet) ser ut, samt hur ett komplett grafikkort i serien ser ut med och utan skyddskåpa.



Figur 4: GT200b-chippet, kretskortet till ett grafikkort och ett färdigt GTX 295-grafikkort. Källa: Nvidia.

3.3 GPGPU

GPGPU-programmering har huvudsakligen inriktats på mycket speciella problem. Detta är naturligt eftersom GPU-hårdvaran inte är lika generell som en traditionell CPU och därför bara är effektiv för vissa typer av beräkningar. En förutsättning för att beräkningar ska passa för GPU-implementation är att de består av en stor mängd instanser av samma problem, det vill säga att samma program kan köras parallellt med olika parametrar. För bästa effektivitet krävs dessutom att de parallella trådarna inte väljer olika väg i de eventuella villkorssatser som finns i programkoden. Detta brukar beskrivas som att det finns en beräkningskärna som ska appliceras på ett eller flera större datablock. Exempel på problem som lämpar sig för GPU-hårdvara är många matematiska operationer (matrismultiplikationer, FFT-beräkningar med flera), sorteringsalgoritmer och viss typ av kryptering.

I takt med att GPGPU-programmering har blivit allt populärare har behovet av ett programspråk för ändamålet vuxit. I februari 2007 släppte Nvidia CUDA (Compute Unified Device Architecture) som är en parallellberäkningsarkitektur för Nvidias GPU:er. Till CUDA kan ett antal olika programspråk användas: C for CUDA som är Nvidias egen utökning av C, OpenCL som är ett royalty-fritt ramverk för att skriva program för heterogena system (till exempel för GPU:er), samt DirectCompute som är Microsofts lösning på samma problem. Även AMD:s grafikkort stödjer GPGPU-programmering via ett eget programspråk. För den som vill läsa mer om GPGPU-programmering finns länkar till mycket information på http://gpgpu.org/developer.

Även med de nya, för ändamålet specialdesignade språken, krävs det en påtaglig arbetsinsats för att implementera beräkningar i grafikkortshårdvara. Befintliga algoritmer, gjorda för standard-CPU:er, måste i allmänhet arbetas om för att fungera effektivt i en GPU. För den som är ny inom området krävs också en inlärningsperiod för att lära sig utvecklingsverktyg och programmeringsparadigm. En fördel med program för grafikhårdvara är att de, om de är rimligt välskrivna, går att flytta mellan hårdvaror med olika prestanda (från samma tillverkare) och kan automatiskt och effektivt dra nytta av den prestanda som finns hos den hårdvara som används vid varje tillfälle. Det innebär att det går att utveckla ett program på en relativt enkel hårdvara och sedan köra det på en mycket kraftfullare hårdvara, och automatiskt få del av den ökade prestandan. Samma sak gäller i viss utsträckning om program flyttas till nyare hårdvarugenerationer.

3.4 Prestanda

Som framgått av de teoretiska värdena på beräkningsprestanda för GPU:er och CPU:er så finns en stor potential att utgå ifrån när beräkningar flyttas till grafikkorten. Den faktiska prestandan för verkliga problem och program kan dock variera stort. För vissa problem kan prestandaskillnaden vara ungefär lika stor som den teoretiska skillnaden. För andra problem, som till synes borde vara lämpade för GPU-beräkningar, kan skillnaden vara endast måttlig. Allt grundar sig i att GPU:er är mycket begränsade i vissa avseenden, och för problem där dessa begränsningar är till hinder kan prestanda bli mycket lidande.

För den som står i begrepp att investera i ett GPU-baserat beräkningssystem finns en väsentlig sak att notera. De grafikkort vi diskuterar i den här rapporten, och alla de som vi använt i våra experiment, är hämtade från Nvidias konsumentserie av grafikkort. Det är alltså i princip spelgrafikkort. Dessa är mycket prisvärda men är inte snabba på att göra flyttalsberäkningar med dubbel precision. Den som är i behov av snabba beräkningar i dubbel precision måste vänta till nästa generation av GPU:er från Nvidia. Värt att notera är att det inte hjälper att stiga upp till proffs/CAD-serien av grafikkort, alternativt de speciella beräkningskort

som Nvidia säljer, Tesla-serien. Dessa innehåller precis samma GPU:er som spelserien, men är ungefär 4–5 gånger så dyra vid en given prestanda. Det snabbaste spelkortet, GTX 295, kostar ungefär 3500 kronor och det snabbaste beräkningskortet, ungefär 12 000 kronor, men då innehåller spelkortet två GPU:er.

Det som skiljer mellan spelserien och proffsserien är byggkvalitet, tekniska marginaler, supportmöjlighet från Nvidia och längden på garantiåtagandet. Spelkorten är byggda för att vara så billiga som möjligt och är ofta pressade till den yttersta gränsen för vad hårdvaran klarar av. Detta innebär att risken för fel blir större, både för räknefel och totalhaveri. När ett grafikkort används för att generera bilder är det inget stort problem om en pixel blir fel. Det finns två miljoner andra pixlar att titta på och den felaktiga pixeln försvinner vid nästa uppdatering av skärmen, 16 millisekunder senare. Om grafikkortet används för beräkningar så kan ett enstaka fel föras vidare i en mycket lång beräkning och omintetgöra ett slutresultat som tagit dagar eller veckor att komma fram till. Det är inte ens säkert att felet upptäcks, vilket förstås är ännu värre. På samma sätt är ett totalhaveri för en hemanvändare irriterande, men inte särskilt problematiskt. Antingen finns det garanti kvar vilket resulterar i ett gratis utbyte, eller så är kortet ändå så gammalt att det kanske är rätt trevligt att byta ut det ändå. I en större beräkningsmaskin med kanske hundratals noder innebär kort totallivslängd på korten att systemadministratörerna ständigt kommer att behöva ägna sig åt grafikkortsbyten.

3.5 Slutsats

För de problem som lämpar sig att köra i grafikhårdvara finns stora prestandavinster att göra. Dagens snabbaste GPU:er är 10–20 gånger snabbare än de snabbaste CPU:erna, och skillnaden ser snarast ut att öka. Det krävs dock en del arbete, dels för att lära sig utveckla för grafikhårdvara, dels för att skriva de program som man är i behov av. I många fall är det dock möjligt att utnyttja program som andra skrivit, eftersom det finns program för en hel del standardproblem att tillgå på Internet, och denna mängd ökar förstås hela tiden.

4 Calculix

I förra kapitlet fann vi att GPU-beräkningar är en lovande metod att på ett billigt sätt få markant högre beräkningsprestana än vad en traditionell CPU kan erbjuda. I syfte att praktiskt testa hur väl lösenfrasforcering i mjukvara fungerar, både med hjälp av vanliga CPU:er och i grafikhårdvara, har vi byggt en beräkningsmaskin som vi kallat Calculix. I det här kapitlet beskriver vi Calculix' hårdvara och mjukvara, samt redovisar de prestandaresultat vi nått vid lösenfrastestning i WPA/WPA2-PSK.

4.1 Beskrivning av maskinen

4.1.1 Hårdvara

Calculix är i grund och botten en standard-PC. Den är byggd på ett moderkort med fyra PCI-express-platser som är tillräckligt glest placerade för att tillåta grafikkort av dubbel tjocklek i alla dessa platser. Detta är monterat i ett chassi med en extra expansionsplatsöppning i bakplåten, för att tillåta även det fjärde grafikkortet att få plats trots att det hänger ned nedanför moderkortets kant. På moderkortet sitter också en AMD Phenom X4 och vanligt standardminne.

PCI-express-platserna är fyllda med Nvidias snabbaste grafikkort, Nvidia GTX 295. Varje sådant består i princip av två separata grafikkort med var sin av Nvidias snabbaste GPU:er. Varje GPU har 240 shader-enheter, vilket ger Calculix totalt 1920 beräkningskärnor som var och en klockas i cirka 1,25 Ghz. I figuren nedan visas hur Calculix ser ut.

Figur 5: Grafikkortsberäkningsdatorn Calculix (till vänster) och dess innandöme (till höger).

4.1.2 Mjukvara

Operativsystemet är en Ubuntu Linux 9.04 Desktop med Nvidias egna drivrutiner för grafikkorten. Själva WPA-PSK-attacken har gjorts med hjälp av programmet Pyrit som är en fri programvara med GNU-licens. Pyrit kan testa lösenfraser för WPA-PSK och kan om man vill även skapa förberäknade databaser med färdigräknade lösenfraser. Pyrit är ett projekt i mycket snabb utveckling. Nya källkodsversioner kommer varje vecka och önskemål och felrapporter från användarna åtgärdas snabbt. Pyrits hemsida hittar man på http://code.google.com/p/pyrit.

Pyrit består av ett ramverk som matar föreslagna lösenfraser till en eller flera beräkningsmoduler. Beräkningsmoduler finns för CPU:er såväl som för grafikkort från både Nvidia och AMD. Beräkningsmodulerna verkar mycket välgjorda. Modulen för CPU:er stödjer SSE2, kan hantera godtyckligt antal CPU:er och CPU-kärnor och är flera gånger snabbare än de motsvarande program vi hittat från andra utvecklare. Beräkningsmodulen för Nvidia-GPU:er är också genomtänkt och stödjer multipla, samtidiga GPU:er, något som är ganska ovanligt i GPGPU-sammanhang eftersom det kräver en del extra arbete i programmeringen.

4.2 Resultat för WPA-PSK-forceringen

Vi har testat olika versioner av Pyrit i ungefär ett halvt år (2009). Utvecklingen under tiden har varit snabb och till exempel har stödet för SSE2 i CPU-modulen tillkommit. SSE2 är, liksom GPGPU-beräkningar, ett sätt att parallellisera likformiga beräkningar. I och med SSE2-stödet, som ökade CPU-modulens prestanda avsevärt, minskade skillnaden i hastighet mellan CPU-modulen och Nvidia-modulen. CPU-modulen klarar numer cirka 3500 nycklar/s på en snabb, fyrakärnig CPU (Intel i7 950) medan Nvidia-modulen hinner med cirka 11 000

nycklar/s på den snabbaste GPU:n. Skillnaden är alltså bara ungefär en faktor tre. Å andra sidan innehåller ett GTX 295 två stycken sådana GPU:er och kostar ungefär lika mycket som en sådan CPU. Calculix, som innehåller fyra GTX 295, är alltså teoretiskt ungefär 25 gånger snabbare än en enskild CPU.

I praktiken har vi inte nått upp till de teoretiska värdena för beräkningsprestanda som ligger på ungefär 80 000 nycklar/s. När vi kör tester på verkliga lösenlistor når vi upp till ungefär 65 000 nycklar/s, det vill säga ungefär 80% av den teoretiska prestandan. För att nå upp till 65 000 nycklar/s krävs dessutom mycket långa lösenlistor, åtminstone några tiotals miljoner lösenord och fraser, för att alla steg i beräkningsprocessen ska hinna fyllas upp och fungera optimalt.

Som kuriosa kan nämnas att det kluster som IT-säkerhetsgruppen vid FOI äger har en prestanda som är närmast identisk med Calculix'. Klustret består av 180 datorer med två CPU:er i varje, där prestandan per CPU är ungefär 180 nycklar/s. Dessa maskiner är fem år gamla, men har dock SSE2-stöd.

5 Lösenlistor

Hittills i rapporten har vi sett att lösentestning är den gångbara metoden för att forcera WPA/WPA2-PSK och vi har studerat hur fort det går att göra sådana lösentester. I praktiken finns också ett behov av välkonstruerade lösenlistor. Vi har provat att tillverka en stor lösenlista för att se hur lätt det är och hur stor den blir. I det här kapitlet redovisar vi hur vi gjort och hur en sådan lista skulle kunna kompletteras ytterligare samt hur listan skulle kunna manipuleras för att hitta ännu fler möjliga lösenfraser.

5.1 Konstruktion av en svensk lösenlista

Eftersom all WPA-PSK-forcering handlar om att testa lösenfraser så har vi samlat indata från ett flertal källor. Vi har valt att göra en svensk lösenlista eftersom det är lättare både att samla material till en sådan och lättare att bedöma hur bra den blir. De källor vi använt är:

- Den stora svenska ordlistan (GPL), 380 000 ord inklusive böjningsformer
- Svenska Wikipedias uppslagsord och uppslagsfraser, 537 000 st
- Svenska förnamn (alla) och efternamn (nästan alla), 270 000 + 150 000 st
- Alla namn på terrängkartan (fjällkartan där terrängkartan saknas), 405 000 st
- Diverse textmassor, 650 000 ord
- Totalsumma: 1 671 340 unika ord och fraser

Det är antagligen inte lätt att göra en sådan här lista så mycket större så länge tonvikten ska ligga på ord från det svenska språket. Det är förstås möjligt att göra utökningar i form av tabeller med olika sorters datamaterial, eller att tillåta olika former av manipulationer och variationer på listans innehåll. Några exempel sådana utökningar ges i uppräkningen nedan:

- Alla tal upp till 9 999 999, med och utan mellanslag, 22 222 222 st
- Alla möjliga bilnummer, tre format, 73 000 000 st
- Den konstruerade listan med upp till en liten förändring, någonstans i storleksordningen 3 000 000 000 st
- Den konstruerade listan med upp till två små förändringar, någonstans i storleksordningen $1{,}5 \times 10^{12}$ st

5.2 Räkneexempel för test av lösenlistor

Efter att ha sett några exempel på hur stora lösenlistor kan bli så är det förstås intressant att se hur tidskrävande det skulle vara att testa alla förslag i dessa listor med hjälp av Pyrit (se avsnitt 4.1.2 om Calculix' mjukvara). För att ge intressanta värden i alla fall så tänker vi oss tre olika beräkningssystem:

1. Ett system med en helt vanlig, snabb CPU. Vi antar att ett sådant system klarar 3500 testade lösen/sekund och kostar 10 000 kronor.
2. Ett system som Calculix. Ett sådant system klarar 65 000 lösen/sekund och vi antar att det kostar 25 000 kronor.
3. Ett system som består av 40 stycken maskiner som Calculix. Ett sådant system klarar 2 600 000 lösen/sekund och vi antar att det kostar 1 000 000 kronor.

Vi tittar vidare på fyra olika storlekar på lösenlistor: 1,5 miljoner lösen (en grundlista med lösenförslag), 3 miljarder lösen (grundlista med en förändring), $1{,}5 \times 10^{12}$ lösen (grundlista med upp till två förändringar) och $2^{56} = 7 \times 10^{16}$ lösen (ett idealt åttateckens lösen, se avsnitt 2.2.1 om säkerhetsmodellen i WPA/WPA2-PSK). Tiden det tar för de olika systemen att testa alla lösenförslagen i lösenlistorna visas i tabellen nedan:

System\liststorlek	**1,5 miljoner**	**3 miljarder**	**$1{,}5 \times 10^{12}$**	**7×10^{16}**
System 1: Vanlig CPU, 3500 nycklar/s	7 minuter	10 dygn	14 år	650 000 år
System 2: Enkelt grafikkorts-system, 65 000 nycklar/s	23 sekunder	13 timmar	9 månader	35 000 år
System 3: Kluster av grafik-kortssystem, 2 600 000 nycklar/s	0,6 sekunder	19 minuter	1 vecka	880 år

Tabell 1: Tiden det skulle ta att testa fyra olika storlekar på lösenlistor med tre olika beräkningssystem.

Värt att notera är att ett slumpmässigt valt lösenord på åtta tecken räcker mycket långt för att säkra ett WLAN-system, såvida inte angriparen har en budget på runt en miljard kronor att lägga på varje nät som ska forceras.

6 Diskussion

Sammantaget har vi sett att WPA/WPA2-PSK principiellt är gynnsamt att attackera för en angripare, men att designen av systemet är gjord med detta i åtanke och därför innehåller skydd som försvårar sådana attacker. Resultatet har blivit ett system som går att angripa genom att testa stora mängder möjliga lösenfraser, men som inte går att angripa genom uttömmande sökning av lösenfraserna annat än om dessa är minimala i storlek och angriparen samtidigt har en mycket stor budget (runt en miljard kronor för varje nät som ska forceras). Säkerheten varierar alltså från helt osäker till helt säker beroende på lösenfrasernas kvalitet, vilket i förlängningen beror på nätägarens kunskap och säkerhetsmedvetenhet.

I de fall nätägaren har valt en svag lösenfras är det i princip enkelt att skaffa tillgång till nätet. Om angriparen lyckas eller inte beror dock på hur välgjord dennes lösenlista är. Att göra en bra lösenlista är antagligen den största utmaningen för en angripare. Rent allmänt är det en mycket intressant fråga hur en sådan ska utformas för att vara så effektiv och fullständig som möjligt. I frågan ligger också vilka manipulationer och variationer på elementen i listan som ska göras för att täcka de fall där lösenfrasen utgår från något som är lätt att komma ihåg och sedan tillförs en liten variation av något slag. En välgjord sådan lista, eller principer för hur en sådan bör se ut, skulle vara värdefullt för en angripare, inte bara i WLAN-fallet utan vid varje angrepp mot ett system som använder lösenord eller lösenfraser i sin säkerhetsmodell.

Vi har sett att även en ganska innehållsrik lösenlista, i sitt grundutförande utan manipulationer, får en ganska måttlig omfattning. Vårt exempel med den svenska listan landade på drygt 1,5 miljoner element. Detta kan även en vanlig dator testa sig igenom på några minuter. Så fort man börjar variera elementen och manipulera dem på olika sätt så växer dock antalet lösenförslag som ska testas och mer kraftfulla beräkningslösningar krävs. Ett grafikkortsbaserat kluster skulle antagligen kunna komma ganska långt i att hitta lösenfraser i de fall nätägaren valt lösenord som är lätta att komma ihåg. Det är inte omöjligt att arbete av en professionell utvecklare skulle kunna öka effektiviteten hos programkoden, och när nästa, eller nästnästa generation av grafikkort dyker upp så kommer en del av dagens begränsningar hos GPU:erna att försvinna och det kommer att bli lättare att skriva effektiv kod. Dessutom blir själva grundprestandan hos GPU:erna snabbare i mycket rask takt.

Vi har också sett att beräkningar på grafikkort är en givande väg i många fall då en vanlig standarddator är för långsam. En förutsättning är att problemet passar för implementation i grafikhårdvara. En annan förutsättning är att arbetsinsatsen med implementationen inte är så stor att vinsten vid beräkningen äts upp av implementationskostnaden och implementationstiden. Återkommande, kostsamma beräkningar är antagligen det som huvudsakligen är relevant att föra

över i grafikhårdvara idag om det inte går att hitta färdiga program. Efter hand som allt fler standardproblem implementeras och publiceras kommer implementationskostnaden för nya program att minska eftersom en större del av koden kommer att kunna hämtas från existerande program eller färdiga bibliotek.

7 Vidare arbete

Det finns några saker som skulle vara intressant att följa upp vidare. En sådan är att vi inte har studerat kvaliteten hos lösenlistan, det vill säga hur ofta den fungerar. Det finns inneboende problem i att göra en sådan studie på ett trovärdigt sätt. Det går inte att fråga folk vad de använder för lösenord och att köra ett test mot en lösenordsdatabas i ett skarpt system är i sig problematiskt och det är tveksamt om resultatet skulle gå att få ta del av. Någon typ av sådant test vore dock relevant att köra. Frågor som skulle vara värdefulla att besvara är:

- hur stor andel av lösenorden väljs så att de är möjliga att gissa?
- vilken typ av gissningsbara ord och fraser väljer folk huvudsakligen (dvs. vilka delar av lösenlistorna ska det läggas särskild vikt vid att förbättra)?
- hur mycket går det att vinna på att använda olika typer av manipulationer för att skapa varitioner av det som finns i listorna?

En annan sak som skulle vara intressant är att studera grafikkortsberäkningar noggrannare. Några frågor som dyker upp är:

- Är de 80% av teoretisk prestanda som vi får i Calculix ett rimligt värde, eller kanske till och med ett bra värde?
- Hur svårt är det att skriva egen kod för grafikkortsberäkningar? När är prestandaskillnaden stor nog för att motivera den extra insats som det innebär att implementera koden för den lite speciella arkitektur som ett grafikkort utgör?
- Vilken prestanda går det att uppnå vid nyckeltestning för vanliga, standardiserade kryptoalgoritmer som AES, DES, A5/x (GSM), RC4 med flera?