Toisenlaiset tietokoneet

Kuvitellaan, että haluat suunnitella tietokoneen, joka on läpikotaisin erilainen kuin yksikään aiempi. Onnistuisiko se?

Teksti: Ville-Matias Heikkilä

Kuvat: Mitol Meerna, Ville-Matias Heikkilä, Wikimedia Commons -käyttäjät Daderot, Shieldforyoureyes, Jitze Couperus, Rama, Rainer Lehrig, D-Wave Systems, Andrei Kulikov

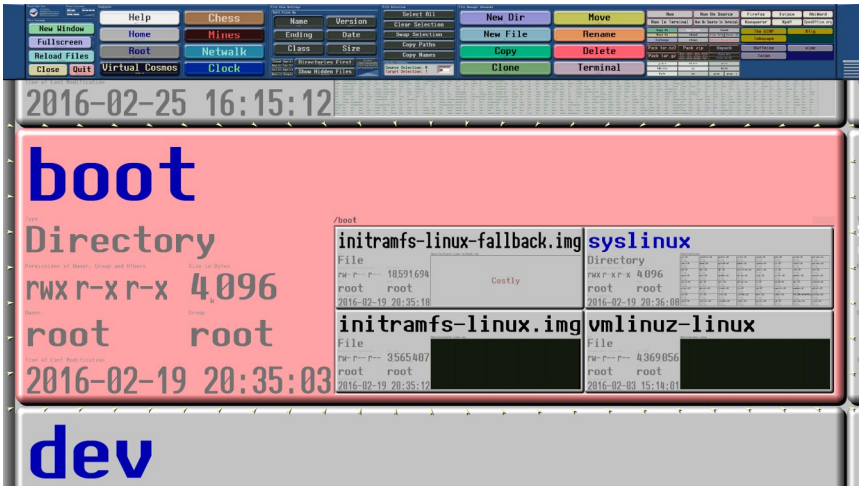
Pyydä tietotekniikkaa tuntematonta kaveriasi kuvittelemaan, millaisia tietokoneita avaruuden muukalaiset käyttäisivät. Todennäköisesti kuvitelman pohjana olisi hänelle tutuin tietokonetyyppi, vaikkapa Windowsia ajava PC, jota olisi muutettu hieman oudomman näköiseksi. Elokuvissa ja tv-sarjoissa jäädään yleensä tälle tasolle: graafisessa käyttöliittymässä voi olla tuttujen kirjainten tilalla hassut vänkyrät ja nelikulmioiden tilalla kolmiot, mutta graafisen käyttöliittymän ideaa ei tule kukaan kyseenalaistaneeksi.

Asiantuntija, joka tuntee monia erityyppisiä tietokoneita eri aikakausilta, pystyy paljon valistuneempaan arvaukseen, mutta hänkin saattaa juuttua kiinni kyseenalaistamattomiin itsestäänselvyyksiin. Tietokonemaailma on nimittäin täynnä vakiintuneita teknisiä ratkaisuja, joiden vakiintuneisuus ei suinkaan tarkoita, että ne olisivat ainoita oikeita.

Ulkoinen olemus

Ulkoisesti erinäköisiä tietokoneita on helppo kuvitella. Periaatteessa mikään ei rajoita sitä, miltä tietokone voi näyttää. Tämän todetakseen ei tarvitse edes mennä mielikuvitusmaailmoihin. Sulautettuja järjestelmiä kun on nykyään lähes kaikkialla pienistä arkiesineistä isoihin rakennuskomplekseihin. Kunhan nanotekniikka kehittyy, tietotekniikan ei tarvitse olla edes olomuodoltaan kiinteää, vaan jokainen epämääräinen kaasupilvikin saattaa olla tietokone.

Jos konetta on kuitenkin tarkoitus pystyä käyttämään tehokkaasti, on joko koneen tai siihen liitetyn päätelaitteen ulkomuodolle hyvä asettaa joi-



Zoomaavia käyttöliittymiä pidetään usein outoina. Sellainen on esimerkiksi Eagle Mode -tiedostonhallintaohjelmassa.

takin reunaehtoja. Useimmat ihmiset tarkastelevat maailmaa ensisijaisesti silmillä ja vaikuttavat siihen käsillä, joten tietokoneissa on ollut sen mukaisesti näyttöruutuja, vilkkuvaloja, kytkinpaneeleita, paperille tulostavia kirjoittimia, virtuaalisilmikoita ja erilaisia osoitinlaitteita. Toimivia vaihtoehtoja näille on melko helppo kuvitella, vaikka pitäydyttäisiinkin pelkissä silmissä ja käsissä. Kaasupilvikoneen nanopartikkelitkin voisivat tuottaa näkyviä kuvioita ja havainnoida käyttäjän liikkeitä.

Kun olemassa olevia käyttöliittymiä tutkitaan syöttö- ja tulostustasoa syvemmältä, voi suurimman osan niistä luokitella joko kädenjatkeiksi tai kielellisiksi. Kädenjatkemallissa manipuloidaan koneen sisäisiä olioita jotakuinkin suoraan. Välissä voi tosin olla jonkinlainen liikuteltava kohdistin tai muu virtuaalihahmo. Kielelliset käyttöliittymät puolestaan perustuvat esimerkiksi komentotulkkeihin tai koneen esittämiin kysymyssarjoihin. Merkittäviä välimuotojakin tosin on, esimerkiksi vi-sukuiset tekstieditorit tai Sierran vanhat seikkailupelit.

Kokonaan jaottelun ulkopuolelle voisi mennä vaikkapa järjestelmä, jossa kone oppii toimimaan käyttäjänsä antaman esimerkin mukaan esimerkiksi neuroverkon ohjaamana. Tämä on suunta, johon peruskäyttäjien ohjelmistot ovat muutenkin hiljalleen menossa; jo nyt ne arvailevat jonkin verran käyttäjän aikeita. Toisaalta jaottelusta putoaa myös esimerkiksi Pure Data -ohjelmointiympäristö, jonka rakenteet eivät perustu kieleen vaan palikoiden graafiseen kytkemiseen toisiinsa.

Tulevaisuuden tietokoneille on yleensä kuviteltu vallitsevan käyttöliittymäfilosofian äärimmilleen kehitetty muoto. Komentorivin valtakaudella oltiin varmoja, että tulevaisuudessa ihmisen ja koneen vuorovaikutus tapahtuisi luonnollisella kielellä, joka voisi olla vaikkapa puhuttua. Nykyisin saatetaan sen sijaan unelmoida käyttöliittymistä, joissa olioita voi muovailla käsillään entistä monipuolisemmin tai joissa kone arvaa entistä älykkäämmin, mitä käyttäjä yrittää milloinkin tehdä. Jos siis käyttöliittymästään haluaa varmuudella täysin erilaisen, kannattaa huomata tällaiset trendit.

Kuoren ja ytimen suhde

Tavallisen käyttäjän käsitys siitä, kuinka kone toimii sisäisesti, perustuu käyttöliittymän toimintaan. Toisinaan pinta onnistuu kuvaamaan sisuksia hyvin, kun taas joskus käyttöliittymä on suorastaan kulissi, joka varjelee käyttäjää käyttöjärjestelmältä. Esimerkiksi älypuhelimet häivyttävät usein käyttäjiltään koko tiedostojärjestelmän hakemistohierarkioineen, vaikka niiden käyttöjärjestelmät olisivat Unix-pohjaisina hyvinkin hakemistopuukeskeisiä.

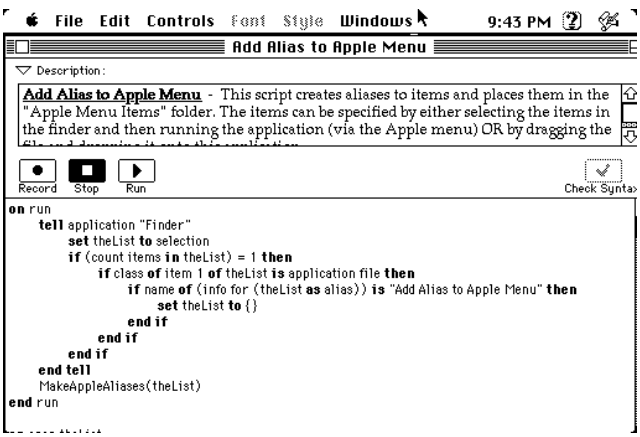
Hyvän harhakuvan täysin uudella tavoin toimivasta tietokoneesta saa siis luotua toteuttamalla käyttöliittymän, joka hämää käyttäjää eri tavoin kuin aikaisemmat. Mutta entäpä, jos ei haluta huijata? Tällöin käyttöjärjestelmän ja käyttöliittymän on toimittava samassa käsitemaailmassa. Näin on esimerkiksi alkuperäisissä Macintosheissa.

Klassisen Mac OS:n tiedostonhallinnassa kuvakkeet ja levyllä olevat tiedostot vastaavat toisiaan yksi yhteen. Tiedoston graafiselle sijainnille on tiedostojärjestelmässä omat kentät samoin kuin tiedostotyypille, joka siis ei missään nimessä ole osa tiedostonnimeä. Myös tiedoston rakenne on poikkeuksellinen. Siinä missä Unix-tyyppiset tiedostot pelkistyvät tavujonoiksi, klassisen Macin tiedostot ovat kaksihaaraisia: datahaara sisältää perusdatan, kun taas resurssihaaraan tallennetaan kuvakkeiden kaltaiset oheissälät.

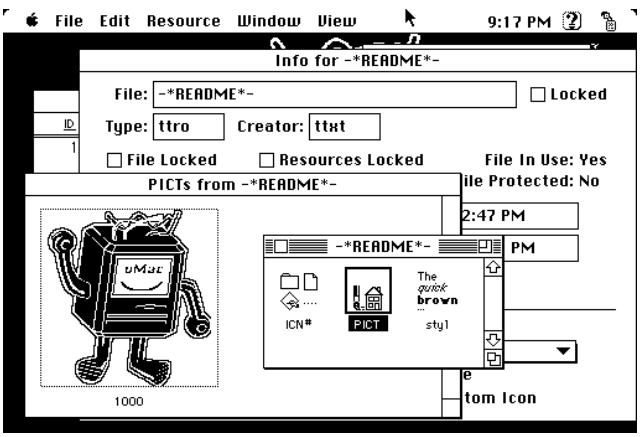
Macintoshin suunnittelijat kyseenalaistivat ahkerasti vanhaa maailmaa. Heille esimerkiksi tekstipäätepohjainen komentorivi oli jäänne vanhanaikaisesta ajattelusta, joten sellaista ei ollut tarjolla edes ohjelmistonkehittäjille. Macintosh Programmer's Workshopin (MPW) komentotulkkia käytetään työarkilta (worksheet), jota voi muokata vapaasti tekstitiedoston tapaan mutta jossa enterin painallus suorittaa kohdistimen kohdalla olevan komennon.

MPW:n kieli muistuttaa Unixin komentotulkkeja. Peruskäyttäjien skriptikielet, kuten Hypertalk ja Applescript, pyrittiin sen sijaan saamaan mahdollisimman luonnollisen kielen kaltaisiksi. Eräässä vaiheessa tavoitteena oli, että Applescriptiä pystyisi lukemaan ja kirjoittamaan samalla kielellä kuin millä muukin käyttöliittymä on. Jos ajatus olisi elänyt pitempään, olisivat suomalaisetkin ehkä päässeet komentamaan Mäkkejään tyyliin ”käske sovellusta 'Microsoft Word' lopettamaan”.

Mac-maailman erilaisuudella oli monia vaikutuksia. Macin käyttäjä pystyi helposti ymmärtämään konetta ja sen virhetilanteita, mutta kun tarvittiin yhteistyötä muunmerkkisten koneiden kanssa, törmättiin oikeaan käsittämättömyyksien ja yhteensopi-



Apple halusi saada skriptien ohjelmoinnista – tai vaikka nauhoittamisesta – mahdollisimman helppoa myös perusjöllille.



Erään Mac-tekstitiedoston sisärakenne avattuna. Teksti on datahaaran puolella, kun taas resurssihaaran puolella on grafiikkaakin.

vuusongelmien muuriin. Kun puolestaan muiden koneiden käyttöliittymät yrittivät matkia Macintoshia, ilmeni omanlaisiaan ongelmia: Amigan Workbench ei tunnusta koko tiedoston olemassaoloa, ellei sen kaverina ole vastaavaa .info-tiedostoa, joka sisältää kuvakkeen ja muut Mac-tyyppiset metatiedot. Windowsit ovat puolestaan usein yrittäneet pimittää tiedostonnimien loppuosat käyttäjiltä, mikä on johtanut sekaviin tilanteisiin.

Nykyään pimittämistä harrastaa myös Macintosh itse. Unix-pohjainen OS X nimittäin vaatii samanlaista kulissia kuin Windowskin, jotta se vaikuttaisi Macintoshilta. Aito, käyttöjärjestelmän ytimeen asti ulottunut erilaisuus korvattiin ”think different” -mainoskampanjoilla, ja pitkään vältelty tekstipäätekin ilmestyi sovellusvalikkoon.

Unohdetut tekoälykoneet

Alkuperäistä Macintoshia voi pitää ideologisena koneena, jonka jokainen osa-alue on suunniteltu tukemaan helppokäyttöisyyttä ja graafisuutta. Mutta se ei suinkaan ole ainoa ideologinen kone. Joskus korkean tason suunnittelufilosofia on ulottunut jopa konekielitasolle asti.

1970-luvun alkupuolella MIT:n tekoälytutkijat tuskastuivat siihen, että heidän suosimansa Lisp-kieli ei oikein taipunut aikakauden koneille. Päätettiin suunnitella työasemakone kokonaan Lispin lähtökohdista. Niin koneen suoritin, käskykanta kuin käyttöjärjestelmäkin rakentuisivat täysin Lispille ja sen filosofialle.

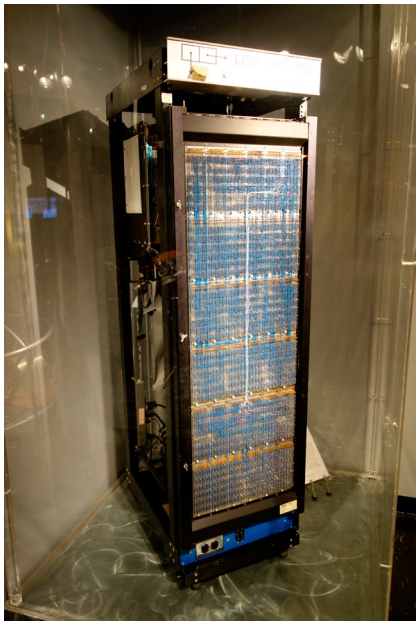
Jos tiettyyn korkean tason ohjelmointikieleen perustuva kone kuulostaa erikoiselta, on hyvä muistaa, että myös valtavirtakoneet on suunniteltu tietyntyyppisille kielille ja ajatteluta-voille. Jos Lisp-koneet olisivat olleet alusta asti valtavirtaa, saatettaisiin meille tuttuja koneita kutsua C- tai Fortran-koneiksi.

Lisp-koneen toiminnassa suurin ero valtavirtakoneisiin nähden on se, että sen data on alas asti tyypitettyä. Jokaiseen muistissa olevaan binäärisanaan kuuluu aina muutama bitti, joilla merkitään, onko kyse esimerkiksi yksinäisestä luvusta vai listaosoittimesta ja mistä listan seuraava alkio saattaisi löytyä. Varsinaisen laskennan rinnalla suoritetaan jatkuvasti omalla piiristöllään tyyppitarkistusta, joka Fortran-koneiden kääntäjäpohjaisissakin Lisp-toteutuksissa vei suurimman osan laskenta-ajasta.

Nykyisin Lisp-koneet ovat varsin tuntematon tietokonehistorian sivujuonne, mutta niiden tulevaisuuteen uskottiin kovasti vielä 1980-luvulla. Lispiä pidettiin vahvana ja ilmaisuvoimaisena kielenä niin tulevaisuuden tekoälysovelluksiin kuin kaikkeen muuhunkin. Koneita valmisti kaupallisesti Symbolics-niminen yhtiö, ja niiden Genera-käyttöjärjestelmä lainasi paljon edistyksellisiä ideoita Xeroxin graafisesta Alto-työasemasta. Nyt Symbolics on ollut konkurssissa jo 20 vuotta, ja konetyyppi on unohdettu voittajien historiankirjoituksesta. Ehkä näkyvin Lisp-koneiden perintö nykymaailmassa on Emacs-tekstieditori.

Lisp-koneet olivat vielä aika maanläheisiä ja tavallisia verrattuna Japanin kauppa- ja teollisuusministeriön, vuonna 1982 aloittamaan ”Viidennen sukupolven tietokonehankkeeseen”, englanninkieliseltä lyhenteeltään FGCS. Hankkeen piti mullistaa koko tietokonemaailma ja laittaa monet tietojenkäsittelyn perusoletukset täysin uusiksi. Laskenta olisi ollut tekoälypainotteista tietämyksenkäsittelyä, jonka perustana oli massiivisesti rinnakkaistettu logiikkaohjelmointi ja ”äidinkielenä” Prolog. Prolog vaihdettiin kuitenkin projektin aikana KL1-nimiseen logiikkaohjelmointikieleen.

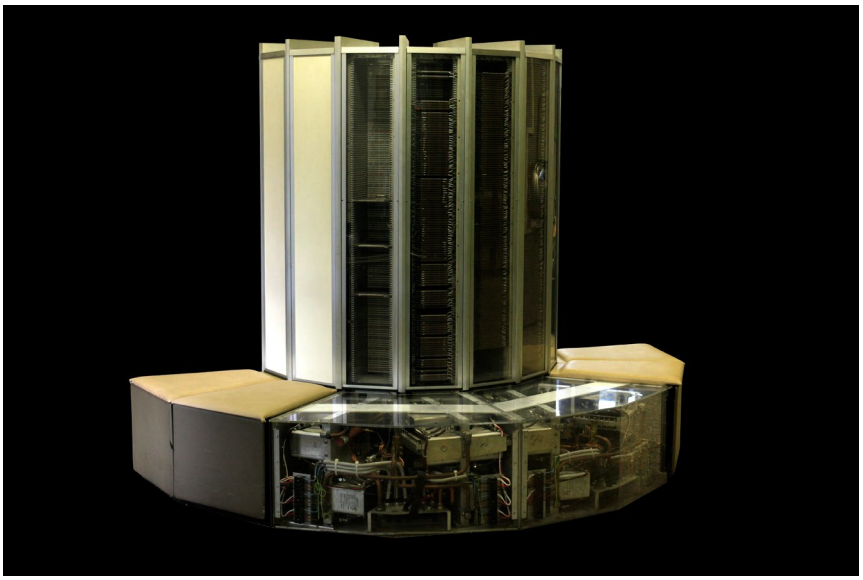
Kymmenen vuoden kehitystyön jälkeen valmistui lopulta viisi erilaista PIM-konetta (Parallel Inference Machine), jotka pystyivät enimmillään muutamaan sataan miljoonaan loogiseen päättelyyn sekunnissa. Yhdysvalloissa samaan aikaan toiminut Thinking Machines oli puolestaan luopunut tekoälykeskeisyydestä jo paljon aiemmin, ja sen Connection Machine -supertietokoneita käytettiin enimmäkseen tavanomaisina supertietokoneina.



CADR oli ensimmäinen sarjatuotettu Lisp-kone.

Symbolicsin Lisp-koneen Space Cadet -näppäimistö: jotain tuttua, jotain outoa.

60-luvun CDC-6600-supertietokone ohjauspöytineen.


Cray-1 vuodelta 1976. Koneen erikoisen muotoilun lähtökohtana ei ole estetiikka vaan komponenttien välimatkojen ja lämmönvaihdon optimointi.

Crayn numeronmurskaushirviöt

Kun puhutaan tietokoneista, jotka tinkimättömästi ilmentävät tiettyä filosofiaa ja näkemystä jokaisella tasollaan, ei kannata unohtaa Seymour Crayta. 60-luvulta 90-luvulle supertietokoneita suunnitellut Cray lähti joka kerta suunnittelemaan konetta, joka murkaisi numeroita nopeammin kuin yksikään aiempi, ja kaikki, mikä ei tukenut tätä päämäärää, pääsi kyseenalaistettavaksi. Hyvin usein Crayn tiimi myös onnistui tavoitteessaan.

Ensimmäinen Crayn supertietokone oli vuonna 1965 valmistunut CDC-6600. Aikana, jolloin valtaosa tietokoneista ajoi ohjelmaa selkeän peräkkäisissä askelissa, 6600:n 60-bittinen keskussuoritin jakoi yksinkertaisia laskentakäskyjään suoritettaviksi useille rinnakkaisille laskentayksiköille. Muut koneet suosivat CISC-ajattelua, jossa yksittäisessä käskyssä oli paljon ilmaisuvoimaa, mutta CDC-6600:n käskykanta oli RISC – parikymmentä vuotta ennen kuin koko termi edes keksittiin. Freonijäähdytteisen laitteiston huippunopeus oli noin kolme miljoonaa liukulukulaskua sekunnissa.

Tietyt periaatteet toistuivat Crayn suunnitelmissa vuosikymmenestä toiseen. Eräs oli se, ettei odottelu ole numeronmurskausraudan arvolle sopivaa, oli se sitten massamuistin, käyttäjän tai syöttö- ja tulostuslaitteiden odottelua. Keskussuoritin piti pitää jatkuvasti työllistettynä, joten vähäpätöisempiin tehtäviin ja keskussuorittimen ruokkimiseen käytettiin erillisiä apusuorittimia ja edustakoneita.

Edes vilkkuvalokonsolien kultaaikana Cray ei laittanut koneisiinsa sellaista, sillä sen käyttöhän olisi vaatinut suorittimen pysäyttelyä. Sen sijaan esimerkiksi CDC-6600:n konsolilla oli kaksi pyöreää vektorikuvaputkea, joille apusuorittimet ajoivat hallintakäyttöliittymää. Aikalaiset varmasti pitivät konetta hirviömäisenä: operaattorit eivät päässeet hallitsemaan elektroniaivoa suoraan, vaan heidän oli keskusteltava mulkosilmäisen välittäjäkoneen kanssa.

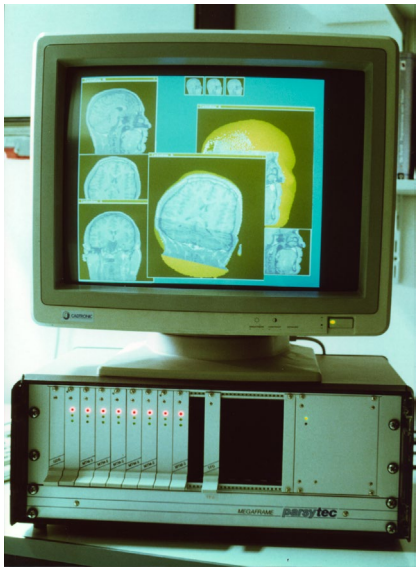
Cray suhtautui yleensäkin nihkeästi ratkaisuihin, jotka olisivat helpottaneet käyttäjän elämää koneen suorituskyvyn kustannuksella. Hyvä esimerkki on näennäismuisti: jos ohjelmoija ei pääse ja joudu itse päättämään, mikä osa datasta on milloinkin keskusmuistissa ja mikä massamuistissa, ei kone yksinkertaisesti voi toimia tehokkaasti. Mikroprosessoreiden käyttöön Cray sentään taipui 1990-luvulla Alpha-suoritinten myötä.

Laskentatapojen moninaisuus

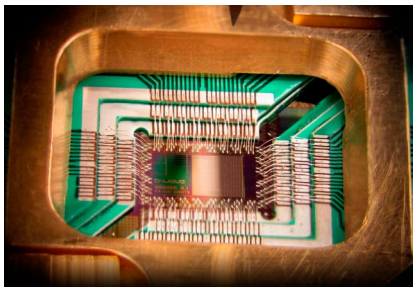
Crayn 60- ja 70-luvun koneissa keskeisenä ideana oli yksisäikeisen peräkkäiskoodin suorittaminen rajoitetusti rinnakkaistaen – kuten on myös useimmissa nykyisissä suorittimissa. Kehitys olisi kuitenkin voinut edetä toisinkin.

Brittiläinen INMOS kehitti 1980-luvulla Transputer-suorittimia, jotka on suunniteltu kytkeytymään nopeilla linkeillä toisiin Transputereihin. Jos Transputerit olisivat yleistyneet, olisi kuluttajatietotekniikassa ehkä päästy kilpailemaan suoritinytimien määrällä jo 10–20 vuotta aiemmin, ja perusohjelmointi olisi alkanut saada rinnakkaista ulottuvuutta jo tuolloin. Transputereita kuitenkin käytettiin lähinnä sulautetuissa järjestelmissä, ja ainoaksi merkittäväksi Transputer-yleistietokoneeksi jäi Atarin flopannut Transputer Workstation. ATW:hen voi asentaa enimmillään 13 kappaletta 20 megahertsin T800-Transputereita.

Nykyisin Transputer-unelmasta on tullut totta: tavallisissakin tietokoneissa on paitsi useita rinnakkaisia suoritinytimiä, myös grafiikkasuorittimia, jotka sopivat erityisen hyvin rinnakkaislaskentaan. Eikä tämä varmasti-



Saksalaisen Parsytecin Megaframe-työasemaan mahtui enimmillään kymmenen Transputer-suoritinta.


D-Waven 128-kubittinen kvanttilaskentapiiri.

kaan jää tähän. On nimittäin odotettavissa, että tulevaisuudessa tietokoneet täyttyvät vielä paljon oudommista laskentayksiköistä. Näitä ovat esimerkiksi ohjelmoitavat logiikkapiirit (FPGA ja rDPA), neurolaskentayksiköt ja kvanttisuorittimet.

Ohjelmoitavan logiikan voi mieltää mikropiireiksi, joiden sisukset ovat sähköisesti vaihdettavissa. Tällaisia piirejä on toistaiseksi käytetty lähinnä korvaamaan ASICeja eli sovelluskohtaisia mikropiirejä, joiden valmistaminen pienellä volyymillä on kallista.

Ohjelmoitavasta logiikasta on kuitenkin enempäänkin. Se nimittäin mahdollistaa ns. rekonfiguroitavan laskennan (reconfigurable computing), jossa laitteisto muuttuu lennossa sopivammaksi kulloinkin käsillä olevalle tehtävälle. Eräs mahdollinen rekonfiguroitavan laskennan malli on Rainer Hartensteinin Xputer. Hartenstein on kutsunut rekonfiguroitavia koneita myös antikoneiksi (”anti-machine”) vittauksena siihen, että niiden ”koneisto” ei ole rakenteeltaan pysyvä.

Neurolaskentayksiköt on tarkoitettu alustaksi ei-biologisille neuroverkoille. Neuroverkkojen vahvuutena on se, ettei niitä tarvitse perinteisessä mielessä ohjelmoida vaan niiden logiikan muodostava painokerroinverkosto rakentuu automaattisesti. Näin ne soveltuvat hyvin esimerkiksi konenäköön, jota on vaikea tyydyttävästi pilkkoa ihmisohjelmoijan käsittämiin moduuleihin. Joidenkin vuosikymmenten päästä jopa ihmismieliä saatetaan kopioida aivoista neurolaskentapiirien ajettaviksi.

Kvanttilaskenta perustuu siihen, että sopivasti rakennettu tietokone pystyy olemaan kvanttimekaanisessa superpositiossa eli useissa tiloissa samaan aikaan. Koneen voi yksinkertaistaen kuvitella laskevan eri asioita eri rinnakkaistodellisuuksissa: jos koneen kvanttibiteistä vaikkapa 8 on superpositiossa, rinnakkaistuu laskenta kaikkiaan 256 eri todellisuuteen. Kvanttilaskenta sopii etenkin tehtäviin, joissa on löydettävä oikea ratkaisu valtavasta mahdollisuusavaruudesta. Kanadalainen D-Wave esitteli ensimmäisen toimivan kvanttitietokoneen joulukuussa 2015.

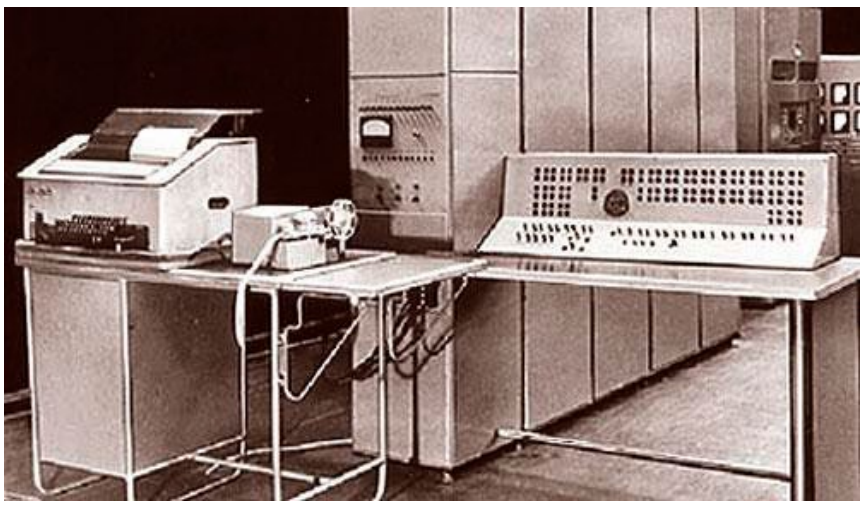
Ykkösen ja nollan tuolle puolen

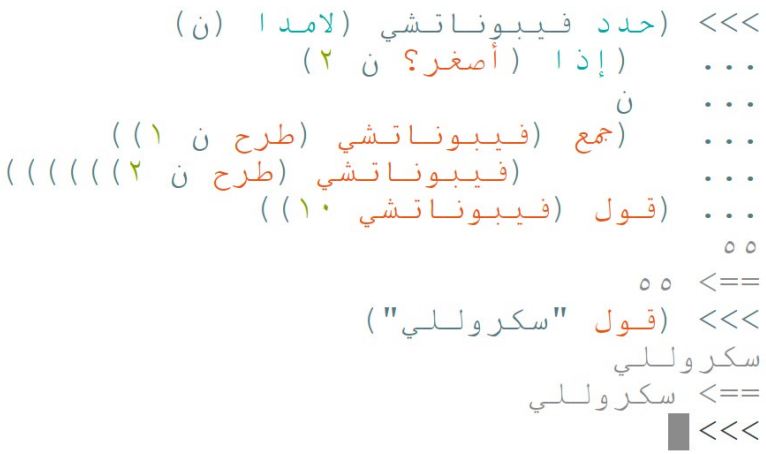
Transputerit jäivät RISCien ja risciytettyjen CISCien varjoon ehkä siksi, että ne vaativat liian suuria ajattelu- ja ohjelmointitavan muutoksia. Valtavirtatietotekniikassa kun uudet paradigmat hyväksytään lähinnä silloin, kun ne saadaan alistettua perinteiselle ajattelutavalle. Vaikka kone olisi täynnä kuinka monimuotoista laskentalogiikkaa, halutaan sitä hallitsemaan tuttu ja turvallinen peräkkäissuoritin, joka ajaa perinteistä käyttöjärjestelmää. Nykyaikaisessa perus-PC:ssä grafiikkasuorittimella on paikkansa omassa karsinassaan, mutta grafiikkasuorittimen hallitsema kone olisi täysin harhaoppinen.

Kun monenkirjavaa joukkoa puhutellaan ylhäältä päin, tuppaa käytetty kieli noudattamaan valtakulttuurin ihanteita. Esimerkiksi sellaiset grafiikkasuoritinpohjaisen rinnakkaislaskennan ohjelmointikielet kuin GLSL ja OpenCL ovat hyvin lähellä perinteistä C-kieltä. Ei siis kannata odottaa, että tulevaisuuden hyperlaskenta johdattaisi meidät kyseenalaistamaan sellaisia tietojenkäsittelyn sitkeitä itsestäänselvyyksiä kuin vaikkapa binäärisyyttä.

Tietokoneet käyttivät jo 1950-luvulla enimmäkseen kolmea eri lukujärjestelmää: binäärisiä kokonaislukuja, binäärisiä liukulukuja ja binäärikoodattuja desimaalilukuja. Moskovan yliopistossa oltiin kuitenkin vakuuttuneita tasapainotetun trinäärijärjestelmän eduista, ja ideaa kokeiltiin siellä niin vuonna 1958 valmistuneessa Setun-koneessa kuin sen seuraajassa, Setun-70:ssä.

Siinä missä binäärijärjestelmä pohjautuu ykköseen ja nollaan, tasapainotettu trinääri ottaa mukaan kolmannen numeron, miinus ykkösen. Etuna on esimerkiksi se, ettei negatiivisia lukuja tarvitse koskaan käsitellä eri tavoin kuin positiivisia ja myös pyöristys- ja kertolaskupiirit saa yksinkertaisemmiksi. Setun saatiinkin pidettyä hyvin eleganttina ja yksinkertaisena, ja sitä tultiin ihailemaan kapitalistimaista asti. Setunit jäivät kuitenkin ainoiksi koskaan rakennetuiksi trinäärikoneiksi,


Setun-trinäärikoneen ohjauspöytä ja muuta laitteistoa.



Qalb on arabialainen Lisp-variantti. Kielen kehittänyt Ramsey Nasser halusi tutkia kirjoitusjärjestelmän vaikutusta ohjelmointikokemukseen ja on tehnyt muutamista ohjelmista jopa kalligrafiateoksia.

vaikka idea palaakin tutkimusartikkeleihin aina muutaman vuoden välein.

Perinteinen Aristoteleen logiikka, johon myös digitaalitekniikassa käytetty Boolen algebra perustuu, on kaksiarvoista: väite voi olla vain tosi tai epätosi. Kaksiarvologiikka ei siis pärjää kovin hyvin esimerkiksi tuntemattomien, ristiriitojen tai sekä–että-tilanteiden kanssa. Jos trinäärikoneet ja niille luontevat ohjelmointitekniikat olisivat yleistyneet, olisivat ohjelmatkin ehkä keskimäärin poikkeussietoisempia.

Tekniikan luonne kumpuaa kulttuurista

Binäärilogiikan voittokulun voi ajatella kuvastavan kulttuurimme sisäistä binäärisyyttä: ”Jos et ole meidän puolellamme, olet meitä vastaan.” Eurooppalaiset kielet ilmaisevat joko–tai-asetelmia huomattavasti selkeämmin kuin monimutkaisempia suhteita, joten meille oli luontevaa kehittää Aristoteleen logiikka, Boolen algebra ja binäärikoneet. Jos tietotekniikka sen sijaan olisi vaikkapa eteläamerikkalaisten aimaroiden kehittämää, saattaisi se hyvinkin olla trinääristä — heidän kielensä kun tarjoaa paremmat rakennuspalikat kolmiarvologiikalle.

Jo pelkkä kieli on siis saattanut vaikuttaa tietotekniikan kehitykseen perustavanlaatuisesti. Meidän on vain vaikea huomata tätä, koska englannin kielen asema on ollut siinä niin vallitseva. Valtaosa ohjelmointikielistä – jopa niistä, joiden avainsanat ovat jollain muulla kielellä – perustuu englannin kielen rakenteeseen. Ohjelmointikielten sanat eivät taivu, ja sanojen suhteet toisiinsa kuvataan tiukalla sanajärjestyksellä.

Suomeen perustuvia ohjelmointikieliä on ainakin yksi: liki unohdettu 80-luvun opetuskieli Sampo. Sen avainsanastoon kuuluu esimerkiksi sijapäätteitä. Läpikotaisin suomen kielen lähtökohdista rakennettu ohjelmointikieli saattaisi siis hyvinkin käyttää lauseenosien suhteiden ilmaisemiseen ensisijaisesti taivutusta ja vasta sen jälkeen sanajärjestystä. Taipuvien sanojen käyttö ohjelmointikielessä on kuitenkin nykyisellään niin outo idea, etteivät edes esoteeristen ohjelmointikielten harrastajat ole juuri pohtineet sitä.

Erilaisten kulttuuripiirteiden vaikutusta tietojenkäsittelyyn voi selvitellä myös vaikkapa repimällä niitä auki postmodernismin hengessä. Hierarkkiset tietorakenteet voivat hyvin olla heijastusta kulttuurimme läpitunkevasta hierarkia-ajattelusta, mustien laatikoiden ylikorostunut asema puolestaan oire teollisen yhteiskunnan vieraannuttavuudesta. Vaikka perusteluja pitäisikin kaukaahaettuina, pystyy tämäntyyppisillä ajatusleikeillä löytämään tekniikasta ja ajattelusta kyseenalaistettavia piirteitä ja etsimään niille mielenkiintoisia vaihtoehtoja.

Tietojenkäsittelytieteilijä Ari Schlesinger esitti pari vuotta sitten ajatuksen feministisestä ohjelmointikielestä, joka olisi rakennettu kokonaan feministisistä lähtökohdista – ohjelmointiparadigmoista ja logiikkajärjestelmistä lähtien. Perinteisen kaksiarvologiikan tilalla olisi kvanttifysiikkainspiroitunut parakonsistentti logiikka, ja olio-ohjelmoinnin sijaan suosittaisiin vähemmän normatiivista abstrahointityyliä. Idean konkreettisempia jatkokehitelmiä odotellaan edelleen mielenkiinnolla.

Visiota kasaan

Tässä artikkelissa käsiteltyjen ideoiden pohjalta saisi varmasti suunniteltua jo aika erikoisia koneita. Miten olisi vaikkapa temppeliä muistuttava kone, jota ohjataan tekemällä sen sisällä erilaisia rituaaleja? Entäpä kone, jonka loogiset perusteet, laskentaparadigmat, ohjelmointikielet ja ohjelmat perustuisivat kaikki sykliselle ajattelulle perinteisen länsimaisen lineaarisuuden sijaan? Tämäntyyppisiä ideoita on mukava pyöritellä mielessään, ja ne toimivat näköaloja avartavina ajatusleikkeinä, vaikkei niitä yleensä kannattaisikaan ruveta konkreettisesti toteuttamaan.

Perustavanlaatuisesti erilaiset tekniset ideat voivat vaatia jopa vuosikymmenten hauduttelua ja työstämistä. Japanin tekoälykoneita kehitettiin isoilla tutkijaryhmillä kymmenen vuotta. Rekonfiguroitavaa tietotekniikkaa ei puolestaan vieläkään oikein ole tarjolla parinkymmenen vuoden tutkimustyöstä huolimatta. Jos siis lähtee kehittelemään jotain täysin uutta, kannattaa oikeasti uskoa visioonsa sen verran, että on valmis heittäytymään ikuisuusprojektiin.

Tietokoneista on vuosikymmenten mittaan tullut aina vain yhdenmukaisempia. Pintapuolisista eroista huolimatta valtaosan sisuksista löytyy enää vain muutamaa erilaista suoritinarkkitehtuuria tai käyttöjärjestelmää, jotka tuputtavat käyttäjilleen aika samantapaisia filosofioita: paljon tuotteistettuja mustia laatikoita, syviä teollisten moduulien hierarkioita, käyttäjän varjelemista tekniikan ymmärtämiseltä. Kysyntää siis varmasti olisi täysin toisenlaisista lähtökohdista kehitetyille tietokoneille – joten siitä vain suunnitteluprojektia käyntiin!