Het vectorgeheugen.

In de meeste multi-programmeringssystemen, die ik gezien heb, moet elk individueel programma zijn geheugenbehoefte opgeven in de vorm "zoveel aaneensluitende geheugenplaatsen". Dynamische wijziging van dit aantal stoot meestal op moeilijkheden -vooral als een programma in de loop van zijn uitvoering tot de ontdekking zou komen, dat het eigenlijk wat meer zou willen hebben.

Mijn opmerking is de volgende: elk programma drukt hier zijn behoefte aan geheugen uit als behoefte aan een vector van gegeven lengte en zodra een aantal van deze programma's samen in eenzelfde geheugen georganiseerd worden, dan beschikt deze overkoepelende organisatie dus over een of ander techniek om een aantal vectoren in een lineair geheugen onder te brengen. Ik zou de koppeling "een individueel, sequentieel programma versus een enkele vector" willen laten vervallen en elk individueel programma de mogelijkheid willen geven, zijn geheugenbehoefte in termen van zo veel vectoren als het maar wil, uit te laten drukken.

Dit heeft enkele potentiele voordelen. Teneerste is het een grote uitzondering, wanneer de behoefte aan geheugen, die door een bepaald proces gesteld wordt, zich op de meest natuurlijke wijze als een enkele vector laat uitdrukken. Men zou, ook bij uni-programmering, van een dergelijke facilitsit veel plezier kunnen hebben. Tentweede schept dit de mogelijkheid tot een wat natuurlijkere eenheid, waarin hoeveelheden informatie van random access geheugen naar, zeg trommel, gedumpt kunnen worden. Tenderde zijn de individuele programma's zo evident niet in termen van physische adressen geformuleerd, dat verwacht mag worden, dat herindeling van het physische geheugen met grotere vrijheid kan geschieden.

In het volgende zal de aanwezigheid van een "tweeslachtig geheugen" (mijn vertaling van "two level store") nauwelijks ter sprake komen. Over de strategie, volgens welke de informatie over beide vormen van geheugen verdeeld zal moeten worden, zal ik het niet hebben; over de administratie van de trommelindeling evenmin.

Mijn voornaamste zorgen zijn voorlopig:
a) dat de programma's in hun formulering -dwz.de manier, waarop ze in het kerngeheugen gerepresenteerd worden- zich beperken tot het "eigene", voor zichzelf wezenlijke; maw. dat deze formulering onafhankelijk is van de wijze, waarop dit programma in zijn omgeving is ingebed.
b) dat de effectieve orde der adressering niet explodeert
c) dat de administratieve taken, die voortvloeien uit de plicht bij herindeling van het geheugen de administratie bij te werken, duidelijk omschreven blijft.
d) dat duidelijk is, welke de "korrelgrootte"is van de programma's, dwz. welke handelingen niet door herindeling van het geheugen verstoord mogen worden. We zouden in practische moeilijkheden kunnen geraken, als de uitvoering van een korrel willekeurig lang zou mogen duren, we zouden een contradictie krijgen, als de uitvoering van een korrel.....herindeling van het geheugen noodzakelijk zou maken!
e) tenslotte een practische zorg: het vinden van een terminologie om hierover te kunnen praten.

Ter vermijding van een babylonische spraakverwarring eerst de volgende afspraak: ik zal het woord "adres" alleen gebruiken in de betekenis van "physisch adres", de bitrij, die het selectieregister van een geheugen ingestuurd kan worden. Elk programma zal zijn elementen identificeren als "nummer zoveel van een vector" en dit rangnummer zal ik een "index" noemen.

Een vector ter lengte N bestaat uit N elementen, onderling onderscheiden door een indexwaarde i; hierbij gelden de ongelijkheden $0 \leq i \leq N-1$. De identificatie van de vector -noodzakelijk indien er in een gegeven context meer vectoren voorkomen- is mogelijk, doordat aan elke vector een zg. "basis" is toegekend en een basis, als element van een andere vector weer door een rangnummer identificeerbaar is.

(De bedoeling is, dat een vector in kerngeheugen een aantal consecutieve geheugenplaatsen zal beslaan; de basis van deze vector bevat dan onder andere beginadres en lengte van de er op steunende vector.)

Het bestaan van een basis is het recht om over een vector te mogen praten; de identificatie (dwz. de indexrij) van de basis fungeert als identificatie van deze vector. Het zal later duidelijk worden, dat een basis -hoewel het een indiceerbaar element van een vector is- niet beschouwd mag worden als een willekeurig element, dat informatie representeert, die beschouwd mag worden als subject matter van het proces. (Een gedeelte van de basis zou zijn het beginadres van de er op steunende vector, maar physische adressen in de context van het programma geen betekenis hebben!) Wij hebben dus twee typen elementen: elementen, waarop het programma vrijelijk opereren mag (getallen) en bases. Om een aantal redenen leggen wij ons de beperking op, dat er geen gemengde vectoren zullen zijn: alle elementen van dezelfde vector moeten van hetzelfde type zijn. Zijn de elementen getallen, dan spreken we zo nodig over een "twijg", zijn de elementen bases van andere vectoren, dan noemen we de vector een"tak". (Een basis bevat, behalve lengte en beginadres, ook een specificatie van het type van de er op steunende vector.)

Elke context veronderstelt de toegankelijkheid van een "impliciete basis", nl. de basis van de vector, waarin de eerste index van een indexrij als rangnummer geinterpreteerd zal worden.

Toelaatbare operaties op vectorbases zijn:
definitie van het type van de er op steunende vector
definitie van de lengte van de er op steunende vector.

Als een bestaande vector verlengd wordt, worden er enige "ongedefinieerde" elementen aan toegevoegd; als het verlenging van een tak is, worden er dus "ongedefinieerde" bases toegevoegd. Een ongedefinieerde basis bevat op de plaats van de type-specificatie de notitie "nog ongespecificeerd". Na deze introductie van de basis mag het type van de er op steunende vector 1 keer gedefinieerd worden. (We sluiten type-veranderingen dus uit voor bestaande vectoren, ook in het geval, dat de vector op dat ogebblik de lengte 0 had.) Een ongespecificeerde vector is iets anders als een lege vector: hoewel er alleen maar "niks" uit zou kunnen komen, heb ik toch bezwaar tegen optellingen als "0 koeien plus 0 ganzen".

Als een bestaande vector verkort wordt, gaat er een aantal van de hoogstgenummerde elementen van deze vector verloren; was de vector een tak, dan gaan met de bases, die afgevoerd worden, alle vectoren verloren, die daar op hebben gerust (en als daar weer takken onder waren, alle vectoren, die daar weer op rusten etc.)

(De in de eervorige alinea genoemde restrictie wat betreft constantheid van type, kan men ondervangen, door als extra operatie op een vectorbasis in te voeren: het ongespecificeerd maken. Zo er al iets op deze basis rustte, gaat dat bij deze ongespecificeerdmakerij verloren. Andere oplossing is, om deze ongespecificeerdmakerij geimpliceerd te laten zijn bij typewisseling. Of dit allemaal erg belangrijk is, waag ik te betwijfelen.)

Operaties op vectorbases geschieden onder opgave van de indexrij, die de basis in kwestie identificeert; is deze indexrij leeg, dan is per definitie de impliciete basis bedoeld. Context zal dus in het algemeen beginnen, de impliciete basis te definieren als basis van een tak van zekere lengte. Daarmee is een willekeurig aantal nieuwe bases geintroduceerd en daarmee is de mogelijkheid geschapen, om nieuwe vectoren te introduceren.

Referentie naar element i van een vector met lengte N, waarbij i niet voldoet aan $0 \leq i \leq N - 1$ is onzin en heeft geen betekenis: het is dan ook de bedoeling dat elke selectie van een vectorelement impliceert, dat er getest wordt, of er aan deze ongelijkheden voldaan is. (Je eist van elk programma een zekere mate van consistentie; is hieraan voldaan, dan heeft de gebruikelijke vorm van "Memory protect" om het programma van meneer Jansen te beschermen tegen de capriolen van meneer Pieterse geen enkele zin meer. Immers, meneer Pieterse beschikt niet meer over de terminologie om meneer Jansen te hinderen.)

Het is duidelijk, hoe elke context op deze manier zijn locale variabelen op uiterst flexibele manier kan invoeren en bespelen. Op de impliciete basis wordt een boompje geplant, dat tijdens zijn bestaan kan groeien en besnoeid kan worden. Nu wordt het echter de hoogste tijd om te zien, hoe onze context met het locale boompje past in zijn omgeving. De impliciete basis, waarover wij gesproken hebben, is nl. geen absoluut machinegegeven: als wij een zo grote stap terug doen, dat wij het gehele geheugen van de machine kunnen overzien, dan is wat intern, dwz. in een gegeven context als impliciete basis fungeert, extern een element, dat zijn plaats in de totale boom heeft en als ieder ander element geidentificeerd wordt door een indexrij van een of andere lengte.

Bovendien: zonder uitzondering kunnen we stellen, dat elke context niet "self contained" zal zijn, maar slechts kan werken bij gratie van voorafgaande specificatie van een aantal parameters. Een programma, dat een van de communicatieapparaten gebruikt, zal dit communicatieapparaat als "formeel" apparaat toespreken; bij activering moet er als overeenkomstige actuele parameter een specifiek apparaat meegegeven worden. Ook dit valt allemaal onder "inpassing in de omgeving".

Om de samenwerking van een gegeven context te bewerkstelligen met grootheden (vectoren, communicatieapparaten), die intern een formele naam hebben en extern een naam, die uitgedrukt is in een terminologie, die intern geen betekenis heeft, heb ik na rijp beraad en lange aarzeling een nieuw soort vector ingevoerd, de zg. "parametervector".

Bepaalde context kan slechts werken, als zij de beschikking heeft over en de impliciete basis en de bijbehorende parametervector. Omdat dit een onafscheidelijk tweetal is, stel ik voor ze samen in dezelfde vector onder te brengen. (De vorige vectoren, takken en twijgen, waren homogeen.We voeren nu dus in de parametertak, een vector met een heel specifieke indeling, bv. het nulde element is de impliciete basis, het eerste element is de heersende waarde van de "opdrachtteller en de overige elementen zijn "parameters. Het zou me niet verbazen, als het begin van de parametervector een heel geschikte bergplaats was voor nog wat meer impliciete "toestandsvariabelen".) Algemeen begint de parametervector met een vast aantal (minstens 1) vectorbases, eventueel een vast aantal elementen voor toestandsvariabelen, en tenslotte een aantal parameters. Als een vector parametervector is, dan is dat in zijn basis vermeld. In het programma wordt naar de formele parameters verwezen door
a) een indicatie, dat het hier een formele parameter betreft; dit selecteert de parametervector;
b) door een index, die de parameter specificeert.

Het idee is, dat een gegeven context A een context B alsvolgt activeert. Context A reserveert een parametervector, vult daarin alle parameters in en laat

en laat dan de machine overspringen naar de nieuwe parametervector annex nieuwe impliciete basis.

Ik wil nu speciaal even nagaan, hoe we een bestaande vector aan een nieuwe context als parameter meegeven. Ik wil dit doen, door op de bijbehorende plaats van de parametervector een copie van de basis van deze vector in te vullen. Dit is heel griezelig, maar voor mijn gevoel moet ik dit doen.

Laten we even teruggaan naar het idee van de impliciete basis: een bepaald programma is geformuleerd in termen van zijn impliciete basis en spreekt zich er niet over uit, hoelang de indexrij is, die in de externe wereld deze impliciete basis volledig identificeert. De selectie tengevolge van een indexrij in specifieke context zou dus betekenen, dat je deze indexrij in gedachten vooraf zou moeten laten gaan door de indexrij, die nodig is om in de omgeving de impliciete basis te vinden. Dat zou betekenen, dat de efficiency, waarmee een programma uitgevoerd wordt afhangt van de afstand tot de wortel van de boom en dat was niet de bedoeling.

Zodra ik echter vectorbases copieer, dan moet ik er wel voor zorgen, dat bij wijziging van deze basis -verschuiving van de er op steunende vector bv.- dit niet alleen in de originele, unieke basis, maar ook tot zijn copieen doordringt.

Het idee van de aparte parametervector is ingevoerd om bij wijziging van een basis te weten, dat je slechts parametervectoren hoeft te scannen om te kijken, of je elementen tegenkomt, die mee gewijzigd moeten worden. Ik geloof, dat dit sneller uitkomt, dan bij elke copiering bij de bron vermelden, waar de copie(en) staat, te meer daar je bij een basis met een enkel bit nog wel bezuiniging kunt bereiken.(Je kunt bij elke basis een "gecopieerd-bit" invoeren. De ene waarde betekent: deze basis is beslist niet gecopieerd, de andere waarde betekent: deze basis is mogelijk wel gecopieerd. Bij elke copiering van een basis vermeld je dit feit bij deze basis. Een basis kan op deze manieren op een gegeven ogenblik een aantal copieen hebben, maar dat tel je niet. Als sommige van deze copieen, door vectorverkorting verdwijnen, dan laat je dat verdwijnen voorlopig onopgemerkt. (Zolang de basis niet gewijzigd wordt, doet de mogelijke copiering er ook niet zo veel toe.) Pas als je de vector gaat verschuiven, kijk je naar de copieringsbit. Als deze staat in de stand "mogelijk een copie", dan onderzoek je de mogelijke parametervectoren, om te kijken, of je wat hebt bij te werken. Vind je inderdaad copieen, dan werk je deze bij, anders zet je de "gecopieerdbit" weer terug.)

Zolang de ene context slechts sequentieel andere contexten activeert -de ene procedure slechts aanroept, als de vorige helemaal klaar is- kan deze activerende context volstaan met slechts een enkele parametervector. Ook voor het geval, dat een bepaalde context wil beschikken over meer parametervectoren, is er, als ik me niet vergis, wel een acceptabele conventie te vinden, zodat in een bepaalde context de parametervectoren gemakkelijk te vinden zijn.

Als je een subroutine aanroept, moet je daarbij vermelden, welke subroutine je dan aanroept. Evenzo wilde ik bij activering van een bepaald programma dit programma meegeven als parameter van het activeringsmechanisme. Ik beschouw een programma als een vector en we spreken bv. af, dat we de basis van deze vector copieren op de plaats van de eerste parameter, voordat we de machine laten overgaan op de nieuwe parametervector.

Tot slot een paar opmerkingen.

Het in parallelle programmering activeren van een nieuw programma kan je beschouwen als een operatie, waarvan het "gewoon aanroepen van een subroutine" een bijzonder geval is. Een van de parameters laat je een seinpaal zijn, die

voor de activering (voor de aanroep) door het buitenste programma op false gezet wordt. Vervolgens activeert het buitenste programma de subroutine en blijft dan hangen op een passering van deze seinpaal. De subroutine eindigt met zijn voltooiing aan te geven door de seinpaal in kwestie op true te zetten, zodat het aanroepende programma pas doorgaat, als de subroutine voltooid is. Na doorgang kan het aanroepende programma deze activering -deze wachtende machine- weer van de master-wachtlijst afvoeren. Ik betwijfel, of dit de meest practische manier is om subroutines aan te roepen, maar het is wel verhelderend, dat het in wezen niets anders is als het toevoegen van een nieuw programma aan de wachtlijst.

Ik ben er in stilte van uitgegaan, dat de aanwezigheid van een vector in de kernen zou impliceren, dat zijn basis zich ook op de kernen bevindt. Liefst hield ik alle takken op de kernen, ook de parametertakken. Parametertakken allemaal op de kernen houden is daarom wel prettig, omdat verschuiving kan impliceren, dat je in parametertakken ver van de wortel nog correcties moet aanbrengen. Als het noodzakelijk is, om parametertakken op de trommel te dumpen, dan is daar overigens nog wel wat aan te doen. Dan moet je bij elke parameter in de tak tevens vermelden, welke indexrij in de context, die de parameter gevuld heeft, deze parameter heeft aangehaald. Dan moet je bij terughalen van de parameterrij van de trommel je realiseren, dat je nu een parametertak van de trommel haalt, dat daarop correcties eventueel niet zijn bijgehouden, maar je hebt dan alle gegevens op kernen, om de parameterrij, die een tijd lang op de trommel "geslapen heeft" weer up to date te maken.

In welke terminologie de wachtende programma's op de wachtlijst van de master onthouden moeten worden is voor mij nog een open vraag; hetzelfde geldt voor de geprogrammeerde seinpalen.

Ik heb vergeten te vermelden, dat ik met opzet het uit te voeren programma als parameter aan het activeringsmechanisme meegeef, en dat ik een dergelijk programma beschouw als een constante vector.(Hier kan je voor zorgen, door in het programma zelf niet de terminologie ter beschikking te stellen, waarmee het zich zou kunnen overschrijven. Onze ALGOL-programma's overschrijven zichzelf ook niet.) Het is nl. heel aantrekkelijk, om een programma, dat output-limited is, in een aantal activeringen tegelijkertijd aan het werk te hebben, zonder het programma meer dan eens op de kernen te hebben staan. (Wat dachten we van het vertaalproces? Of het "consoleprogramma" als je meer dan een console hebt?)

De grondgedachte bij de kernenbezetting is, dat een vector of helemaal wel, of helemaal niet op de kernen te vinden is. Dat noteer je dan in de basis. Dat je op deze manier geen vectoren in zou kunnen voeren, die langer zijn dan de omvang van het kernengeheugen, is een hinderlijkheid, die echter automatisch op te lossen is (door een impliciete ordeverhoging; dat dit gebeurd is, kan je eveneens in de basis aangeven).

Tenslotte: ik heb zo'n gevoel, dat de transporttijden van en naar langzaam geheugen wel eens een bottle neck zouden kunnen worden. Als je een vector uit het langzame geheugen hebt overgenomen, kan je met een bit in de basis aangeven, dat de copie nog "gaaf" is; schrijf je in die vector, of verander je zijn lengte, dan zet je die bit in de andere stand. Op die manier kan je je vrijwat nodeloos terugtransporteren wel besparen. Als de tijd tussen aanvraag en begin van de feitelijke transport lang is vergeleken bij de feitelijke transporttijd (omdat de trommel in de verkeerde stand staat) kon het wel eens de moeite lonen, om aangevraagde transporten even te verzamelen en desnoods via programma te sorteren. Dit kon wel eens zoveel te meer noodzakelijk zijn, omdat een heleboel vectoren kort zullen zijn vergeleken bij de capaciteit van een heel trommelspoor.

Slotopmerkingen.

Ik ben mij maar al te goed bewust, dat in het voorafgaande geen afgerond plan is beschreven. Integendeel: dit laat nog zoveel vraagtekens over, dat uitwerking een considerabele hoeveelheid werk zal zijn. Voordat ik mij daar in stort, wil ik graag de opinie van wat andere mensen hebben, hoe uitvoerbaar of hoe heilloos dit hen voorkomt.

Ik kan me voorstellen, dat de lezer het gevoel heeft een kijkje achter de schermen van andermans nachtmerrie gekregen te hebben. Het misschien niet overtuigende verweer, dat ik daartegenover stellen kan, is, dat toen het ALGOL-complex voor de X1 in zijn huidige vorm begon te dagen, ons de eerste drie weken ook de rillingen over de rug liepen. Het kan achteraf wel eens heel goed geweest zijn, dat wij slechts etappesgewijze met de consequenties van die aanpak geconfronteerd zijn geworden; anders had het ons meer moeite gekost de euvele moed op te brengen. In mijn niet zeldzame momenten van twijfel put ik wel moed (of moet ik zeggen: overmoed?) uit die ervaring.

Er is nog een reden, waarom ik dit plan nu graag aan anderen wil voorleggen. Van het hele veld problemen, heb ik er een paar uitgekozen en er over gedacht, hoe je die zou kunnen oplossen. Ik heb me daar alleen maar in kunnen verdiepen, door er maar eens van uit te gaan dat de rest wel oplosbaar is. Je hoopt natuurlijk, dat in de keuze van waar je eerst naar kijkt, je intuitie je niet bedriegt, maar je loopt natuurlijk altijd de kans, dat je een paar heel wezenlijke moeilijkheden onbewust naar achteren schuift omdat je er geen verstandig woord over weet te zeggen.

Vandaar, dat dit stuk besloten wordt met een verzoek om commentaar.

E.W.Dijkstra