Ideas this is a counter-proposal to

- Oil: adding a second fuel source to the EVM (pre-EIP)
- https://gist.github.com/holiman/8a3c31e459ee1bff04256bc214ea7f14

Background

Currently, there are many reasons why we want to increase the gas cost of many operations. Most particularly, we want to massively increase (eg. by 3-10x) the gas cost of BALANCE, EXT*, SLOAD and other operations that access state, both because we want to bound witness size and because we want to bound the harm that could be caused by known DoS attack vectors. However, increasing gas costs is dangerous, because contracts sometimes call other contracts with fixed gas limits, and so if gas costs are increased by a hard fork, applications that worked before may cease to work.

The oil/karma proposals linked above solve this by introducing two classes of gas, “gas” and “oil” (or “karma”). Gas works as before, and gas costs of operations are never changed upward. Any new/increased gas costs are instead added as oil costs. A transaction aborts if the total (gas+oil) spending exceeds the total tx gaslimit, and the gas payment is based on gas+oil spent. Importantly, if A calls B, A can restrict how much gas B spends (as today), but A cannot restrict how much oil B spends

.

The proposal solves the problem, but IMO does so at the expense of a great increase in complexity and is very inelegant in how it solves the problem, because one would still somehow need to bound how much oil the child call burns if it is untrusted (see below).

Stepping back: why do we even need gas-bounded subcalls?

One simple fix to this problem is to make all

gas into oil; that is, when A calls B, make it so that A is always

forced to entrust B with the ability to spend all remaining gas

. And it turns out that most applications already work this way

; there are relatively few exceptional cases where A assigns a limited quantity of gas to B.

There are two cases:

1. The 2300 gas limit baked into ETH-sending transactions, intended to allow a log to be generated
2. Cases where A does not trust

B

(2) is surprisingly rare; the main example of this used today is “meta-transactions”, where Bob wants a call to be made to B but has no ETH, so Bob signs an authorization that allows Alice to make a call that (i) calls B and (ii) withdraws some ERC20 token from Bob to Alice. The contract A that implements this procedure needs to

limit gas allocated to B, because otherwise Bob could grief Alice by constructing a B that consumes all the gas, preventing the ERC20 token transfer in the second half of the transaction from being made.

My counter-proposal aims to provide alternate solutions to both (1) and (2). I will focus on (2) first.

The proposal

- Option 1

: Allow a transaction

to specify a table T: {account: gaslimit}

. A call to an account A

in T

would only be assigned a maximum of T[A]

gas, and otherwise calls would be assigned all the remaining gas.

- Option 2

: Allow a transaction to specify a parameter 0 <= r <= 63

(representing the fraction r/64), where if an EVM instance has X

remaining gas and it makes a call, the call would only be given X * r // 64

gas, so the parent would still be guaranteed to have X * (64-r) // 64

gas remaining

- Option 3

: allow a transaction to specify a parameter M

; the gas given to all child calls is multiplied by M

- Option 4

: just change the 63/64 constant introduced inhttps://github.com/ethereum/EIPs/issues/114 to 3/4

The intent of options (1, 2, 3) is to move the choice of how much gas to give to child calls to the transaction sender, allowing the transaction sender to specify how much they trust the accounts that are being child-called into. Option (4) is a “dumber” but simpler fix; it ensures that in any

call there will always be enough gas to do many things both in the child call and after the child call, and any gas cost increase can be dealt with by the transaction sender simply upping the global gas limit.

Alternatives to the 2300 gas minimum

The 2300 gas minimum was originally put into place to ensure that if an account receives ETH it can log this fact; this is useful for eg. wallets. And indeed, 2300 gas is not sufficient to do anything but issuing one or two logs. I propose two options that solve this problem but avoid enshrining a single gas limit:

- Option 1

: the transaction sender specifies the gas minimum, and to avoid introducing re-entrancy issues we add a rule that an ETH-transferring call that provides 0 additional gas does not allow the child call to make any state changes (ie. it’s similar to STATICCALL except it can log)

- Option 2

: we remove the 2300 gas entirely (or don’t touch it), and instead add an explicit log every time any account receives ETH for any reason (including the currently not-covered case, self-destructs).